Threads: Consultando dados de um banco sem comprometer sua aplicação

Clube Delphi

Quick Update
Delphi e Silverlight

Expert
Bug Tracking e API Hooking

Ano 9 :: Edição 127 :: R$ 11,90

DevMedia

Cloud Computing no DELPHI XE

ISSN 1517990-7

Brinde

Comprando esta edição ganhe 2 vídeo-aulas:

- AcBr - Introdução à NFe
- Delphi Prism – Desenvolvimento Web: Cache de página

Visual Studio vs. RAD Studio
Comparativo entre os IDEs, para desenvolvedores Delphi Prism – Parte 1

Delphi Prism
Conheça os melhores recurso da linguagem – Parte 1

Dificuldades em encontrar conteúdo sobre .Net para iniciantes?

A DevMedia simplifica pra você!

easy .net magazine

Easy .Net Magazine:

A primeira revista digital feita especialmente para quem está dando os primeiros passos em .Net!

Você encontrará exemplos e modelos fáceis e práticos, noções introdutórias e conceitos básicos.

Conheça as vantagens:

- Vídeos e apresentações "embutidos" nas páginas;
- Ilustrações, diagramas e modelos apresentados em PowerPoint;
- Entrevistas em vídeo;
- Códigos-fonte que podem ser copiados com CTRL+C/CTRL+V.

A Easy.Net não é feita em PDF ou Flash!
Ela utiliza um aplicativo 100% online.

DevMedia group

Acesse agora a edição Nº 1 totalmente grátis e comprove

www.devmedia.com.br/easynet

Easy.Net Magazine, seu curso inicial de .Net!

Sumário

Quick Update

06 – Delphi e Silverlight
[*Guinther Pauli*]

Delphi

Boa Ideia

10 – Banco de Dados com Threads
Utilizando threads para consultar dados de um banco sem comprometer sua aplicação
[*Marcos Aurélio Rocha*]

Novidades - Mini-Curso

16 – Delphi Prism
Melhorias da linguagem – Parte 1
[*Rodrigo Araujo*]

Boas Práticas

Boas Práticas – Boa Ideia – Expert

24 – MadShi Collection
Tratamento de exceções e hooking de API e código
[*Vitor Luiz Rubio*]

Boas Práticas – Novidades

34 – DataSnap e Windows Azure
Cloud Computing no Delphi XE
[*Fabrício Hissao Kawata*]

Easy Delphi

Easy Delphi – Mini-Curso

42 – Visual Studio vs. RAD Studio
Comparativo entre os IDEs, para desenvolvedores Delphi Prism – Parte 1
[*Rodrigo Araujo*]

Olá, eu sou o **DevMan**! Desta página em diante, eu estarei lhe ajudando a compreender com ainda mais facilidade o conteúdo desta edição. Será um prazer contar com sua companhia! Confira abaixo o que teremos nesta revista:

Brinde na web desta edição - 2 vídeos! Confira:

2 Vídeos

- **Curso NFe com AcBr: Introdução à NFe**
  http://www.devmedia.com.br/post-18842-Introducao-a-NFe-Curso-NFe-com-Delphi-e-AcBr-Parte1.html

- **Curso Delphi Prism – Desenvolvimento WEB: Cache de página**
  http://www.devmedia.com.br/post-18906-Cache-de-Pagina-Desenvolvimento-Web-Curso-Completo-Delphi-Prism-Parte-81.html

Para acessar os vídeos utilize os seguintes dados
Login: **DVM.PL** *Senha:* **GJJ1A22**

Gostou das vídeo aulas? O portal *www.devmedia.com.br* possui mais de **2 mil vídeo aulas** e **dezenas de cursos online** sobre desenvolvimento de software! Agora você pode comprar as vídeo aulas que preferir e fazer sua própria combinação de vídeos! Saiba mais em *www.devmedia.com.br/creditos*

Dê seu feedback sobre esta edição!

A ClubeDelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre esta edição, artigo por artigo, através do link:
www.devmedia.com.br/clubedelphi/feedback

Para votar, você vai precisar do código e senha de banca desta edição, que é:
Login: **DVM.PL** Senha: **GJJ1A22**

Clube Delphi

Ano 9 - 127ª Edição 2011 - ISSN 1517990-7 Impresso no Brasil

Corpo Editorial

Editor Geral
Guinther Pauli
guinther@devmedia.com.br

Editor Técnico
Paulo Quicoli
pauloquicoli@devmedia.com.br

Daniel Laporte
daniel.sobrinho@devmedia.com.br

Equipe Editorial
Vitor Luiz Rubio
vitorrubio@gmail.com

Rafael Stavarengo
faelsta@gmail.com

Jornalista Responsável
Kaline Dolabella - JP24185

Capa e Diagramação
Romulo Araujo - romulo@devmedia.com.br

Revisão
Caroline Velozo - carolinelopes@devmedia.com.br

Coordenação Geral
Daniella Costa - daniella@devmedia.com.br

Distribuição
Fernando Chinaglia Dist. S/A
Rua Teodoro da Silva, 907
Grajaú - RJ - 206563-900

Atendimento ao Leitor
A DevMedia conta com um departamento exclusivo para o atendimento ao leitor. Se você tiver algum problema no recebimento do seu exemplar ou precisar de algum esclarecimento sobre assinaturas, exemplares anteriores, endereço de bancas de jornal, entre outros, entre em contato com:

Isabelle Macedo e Uellen Goulart – Atendimento ao Leitor
www.devmedia.com.br/mancad
(21) 3382-5038

Kaline Dolabella – Gerente de Marketing e Atendimento
kalined@terra.com.br
(21) 3382-5038

Publicidade
Para informações sobre veiculação de anúncio na revista ou no site e para fechar parcerias ou ações específicas de marketing com a DevMedia, entre em contato com:

Cristiany Queiróz

publicidade@devmedia.com.br

EDITORIAL

Houve um tempo em que cada um tinha o seu próprio poço, de onde vinha a água para o consumo diário da casa, assim como também houve o tempo em que as pessoas precisavam de grandes geradores de energia movidos a diesel para ter energia elétrica em suas fazendas. Hoje, tanto a água quanto a energia elétrica vêm até nós, através de canos e fios, abastecer nossas casas e dar conta das necessidades diárias que temos desses recursos. Você por acaso sabe o caminho que a energia elétrica percorre até chegar à sua casa? Todo o processo que envolve essa geração e abastecimento, desde o princípio lá na usina (provavelmente hidroelétrica) até o ponto de energia onde você liga o seu PS3? E a água, você sabe todo o percurso que ela faz, desde os rios, passando pelas estações de tratamento, até chegar aí na torneira da pia do seu banheiro? Não, a maioria de nós não tem nenhuma pista de como tudo isso funciona, e também não damos a mínima para isso. O fato é que consumimos água e energia elétrica, além de muitos outros recursos, sem nos preocupar de onde vem e como foi chegar ali. Vamos trazer este cenário para o desenvolvimento Delphi. O que a Microsoft está oferecendo com o Windows Azure, agora suportado no Delphi XE, não é apenas um provedor gigante para hospedar suas aplicações, ela está oferecendo recursos de processamento e armazenamento, como se fosse energia elétrica ou água potável. São enormes data centers distribuídos pelo mundo, no que podemos literalmente chamar de fazendas de servidores, formando uma rede enorme de hardware. Nessa gigante teia de servidores roda um sistema operacional diferente, arquitetado especialmente para funcionar em nuvem, e o qual foi chamado de Windows Azure. E é dessa enorme usina de processamento e memória que vêm os recursos que consumimos. Quando usamos uma aplicação que está publicada no Windows Azure, estamos consumindo processamento e memória dessa enorme estrutura. A forma como todo esse processamento e armazenamento é cobrada de você também segue um princípio muito parecido com o dos serviços de energia elétrica e água. É basicamente de acordo com o seu consumo que será calculado o valor da sua conta. Isso tudo nos abre outros horizontes, não só na forma em como construímos nossas aplicações, mas principalmente na forma como nós vamos oferecê-las aos nossos usuários, ou deveríamos passar a chamá-los de consumidores? Nesta edição, você vai conferir em primeira mão a integração do Delphi XE com a plataforma Windows Azure para Cloud Computing, no artigo do Fabrício.

Threads é o tema do artigo do Marcos, que demonstra os fundamentos da programação paralela no Delphi, com um exemplo envolvendo banco de dados. O Rodrigo Araujo traz dois artigos nesta edição, iniciando dois mini-cursos voltados aos desenvolvedores que utilizam o Delphi Prism. Domine o IDE do Visual Studio reaproveitando seus conhecimentos no RAD Studio. Além disso, domine as melhorias na linguagem ("D#"). Confira também a matéria do Vitor, que apresenta uma das bibliotecas mais poderosas para programação de baixo nível no Delphi, usufruindo de técnicas de API Hooking e Bug Tracking.

Guinther Pauli
guinther@devmedia.com.br
Certificado Delphi 3,5,6,7,2005,2006,Web,Linux
Microsoft Certified: MCP,MCAD,MCSD.NET,MCTS,MCPD (ASP.NET,C#)
guintherpauli.blogspot.com - twitter.com/guintherpauli

Fale com o Editor!

É muito importante para a equipe saber o que você está achando da revista: que tipo de artigo você gostaria de ler, que artigo você mais gostou e qual artigo você menos gostou. Fique a vontade para entrar em contato com os editores e dar a sua sugestão!
Se você estiver interessado em publicar um artigo na revista ou no site ClubeDelphi, entre em contato com os editores, informando o título e mini-resumo do tema que você gostaria de publicar:

Guinther Pauli - Editor da Revista
guinther@devmedia.com.br

ClubeDelphi PLUS

DevMedia

A revista ClubeDelphi é parte integrante da assinatura ClubeDelphi PLUS.
Para mais informações sobre o pacote PLUS, acesse:
http://www.devmedia.com.br/clubedelphi/portal.asp

Sistema de créditos DevMedia

Conteúdo **completo** ao seu alcance

Com o sistema de créditos DevMedia você pode comprar individualmente mais de 4.000 vídeo aulas, 8000 artigos, além de palestras e vídeos com formações completas.

TODAS as vídeo aulas que você sempre quis ver a partir de

R$ 0,99

ECONOMIA: você não precisa assinar 8 produtos diferentes para ter acesso ao conteúdo completo do site!

Saiba mais sobre o Sistema de Créditos!

www.devmedia.com.br/creditos

DevMedia

Delphi e Silverlight

O Delphi 3 trouxe uma tecnologia muito interessante chamada ActiveX Forms. A ideia era criar formulários Delphi que eram compilados em forma de DLL / OCX no padrão ActiveX / COM, que podiam ser então embutidas em páginas HTML com WebBroker. O resultado eram forms ricos embutidos (embedded) dentro de páginas, com interatividade muito maior que as páginas pobres e estáticas da época. Não era necessário conhecer detalhes sobre desenvolvimento Web, desenvolvíamos como na VCL, e num passe de mágica lá estava nossa aplicação dentro de um browser. A tecnologia morreu por vários motivos, a principal foi obviamente questões de segurança. Mais de 10 anos depois eis que surge uma ideia bastante semelhante.

O .NET Framework 3.0, que não culminou com o lançamento de uma versão específica do Visual Studio, trouxe inúmeras novidades. Tivemos basicamente quatro grandes pilares, chamados respectivamente de Windows Communication Foundation (WCF), Windows Presentation Foundation (WPF), CardSpace e Windows Workflow (WF). Estes na verdade eram frameworks construídos sobre a versão 2.0 do .NET, na época a Microsoft tinha em mente chamar este release de WinFX, porém devido a várias divergências na comunidade internacional, a equipe voltou atrás e o batisou de .NET Framework 3.0. Foi sobre estes pilares que foi construído o Windows Vista, e mais tarde o Windows 7.

Um dos pilares do .NET Framework 3.0 foi o WPF, como já disse, basicamente uma tecnologia que unifica a forma como desenhamos interfaces de usuário, com uso de recursos avançados de processamento gráfico. O WPF utiliza como base uma linguagem chamada XAML - eXtensible Application Markup Language, uma linguagem declarativa baseada em XML. A ideia por trás do WPF é que muitas das funcionalidades e comportamentos de uma interface, como por exemplo, animações e efeitos, pudessem ser feitos também de forma declarativa, como os elementos gráficos, ao invés de usar código programático como C#.

Fazendo um comparativo a grosso modo, imagine um arquivo XAML como se fosse um arquivo DFM da VCL do Delphi Win32. É no DFM que estão definidos os elementos gráficos de um formulário. No código PAS, temos o comportamento, como implementação dos eventos, acesso a dados etc. Da mesma forma, no WPF também temos um arquivo associado ao XAML, normalmente chamados de code-behind (mesma ideia do ASP.NET), que contém a lógica de programação. Para ficar mais claro, veja o código XAML da **Listagem 1**. Nela temos um botão e um TextBox. Para simplificar, não vamos considerar questões como layouts e posicionamento.

Apesar de ser mais indicado para construção de aplicações que exigem interfaces mais rebuscadas e arrojadas, como as que rodam em dispositivos com reconhecimento a toque, como o Microsoft Surface, muitos desenvolvedores começaram a utilizar a tecnologia para o desenvolvimento de aplicações Desktop, inclusive comerciais. Isso porque o framework possui bem mais recursos visuais que o seu ancestral Windows Forms, a tecnologia até então usada para desenvolvimento Desktop no .NET. Nesse sentido, o WPF é concorrente direto da VCL Win32 do Delphi.

Listagem 1. Exemplo de código XAML

```
<Grid x:Name="LayoutRoot">
  <Button Content="Olá Mundo" Name="button1"/>
  <TextBox Text="Delphi Prism!"/>
</Grid>
```

Depois do lançamento do WPF, tivemos uma versão beta do produto chamada WPF/E, cujo objetivo era possibilitar que essas aplicações desenvolvidas em WPF para Windows pudessem rodar dentro de um browser, na forma de um plug-in. Nascia aí o Silverlight.

Silverlight é um framework para desenvolvimento RIA – Rich Internet Application, baseado no WPF / XAML. É também um plugin cross-browser, cross-platform, semelhante ao Adobe Flash, que permite executar "mini aplicativos" dentro de um browser (semelhantes aos ActiveX Forms que abriram este Quick). Veja que, apesar de rodar dentro de um browser, o conceito é bem diferente de uma aplicação Web. Aplicações Web, como as feitas com ASP.NET e AJAX, geram código HTML puro, com JavaScript e CSS. Nada de plug-ins ou instalações adicionais. Isso roda em praticamente qualquer dispositivo que tenha um browser (PC, Mac, SmartPhones (iPhone), PlayStation etc.). Obviamente que no lado servidor vamos precisar do Windows / .NET, a partir de agora certamente o Azure. Já para rodar o Silverlight, é necessário instalar o runtime (plugin) no browser, o que tira da jogada alguns dispositivos. Existem inúmeras especulações a respeito disso, inclusive vindas diretamente da Microsoft, sobre o futuro do Silverlight, muito tendo em vista a concorrência com o HTML 5 e CSS 3, que são padrões abertos. Não vou entrar nestes detalhes aqui, mas deixo algumas dicas na sessão links. O objetivo aqui é mostrar como usar o Silverlight no Prism.

O fato é que hoje o Silverlight roda em várias plataformas e browsers, incluindo obviamente o IE, Google Chrome, Firefox, além de rodar em outros sistemas como

Mac (Safari) e Linux, graças a implementação do Silverlight para a plataforma Mono, o chamado projeto Moonlight. Além disso, o Silverlight é a tecnologia padrão para desenvolvimento para o Windows Phone 7, plataforma que entra no mercado para concorrer diretamente com o iOS da Apple (que roda no iPod, iPhone e iPad) e Android da Google.

É claro que desenvolver com Silverlight é bem mais simples do que desenvolver aplicações Web "nativas", principalmente para desenvolvedores Desktop, como programadores Delphi, o que tem despertado a atenção de muitos. Ganha-se de um lado, perde-se de outro, a questão de escolha depende muito das circunstâncias e objetivos. O fato é que hoje é possível desenvolver aplicações WPF e Silverlight usando Delphi, através do Prism no Visual Studio, e isso nos abre um leque muito grande de opções. Neste Quick Update vamos inclusive fazer um exemplo prático ("Olá Mundo"), a seguir.

O Silverlight é um subset do .NET Framework. De fato, não é necessário instalar todo o .NET Framework na máquina cliente (browser), apenas alguns poucos megabytes. O desenvolvimento da aplicação segue o modelo drag & drop da VCL. A **Figura 1** mostra o template que deve ser escolhido para criar uma aplicação Silverlight no Delphi Prism XE / Visual Studio 2010. Nesse ponto surge uma outra janela (**Figura 2**), onde são informadas as opções para o projeto. Podemos observar que são criados dois projetos, a aplicação Silverlight em si, e uma aplicação Web para hospedar o componente compilado (que pode ser ASP.NET Web Site, Web Application ou MVC).

Veja que nesse momento surge um editor visual no designer do Visual Studio, onde você pode desenvolver de forma muito semelhante ao que está acostumado a fazer com a VCL. Para este exemplo vamos arrastar a partir da Toolbox um Button e um TextBox, como mostra a **Figura 3.** Repare que o código XAML vai sendo gerado logo abaixo, e a ferramenta é two-way, no velho bom estilo do IDE do Delphi. Mude o código diretamente no XAML e veja o resultado no designer.

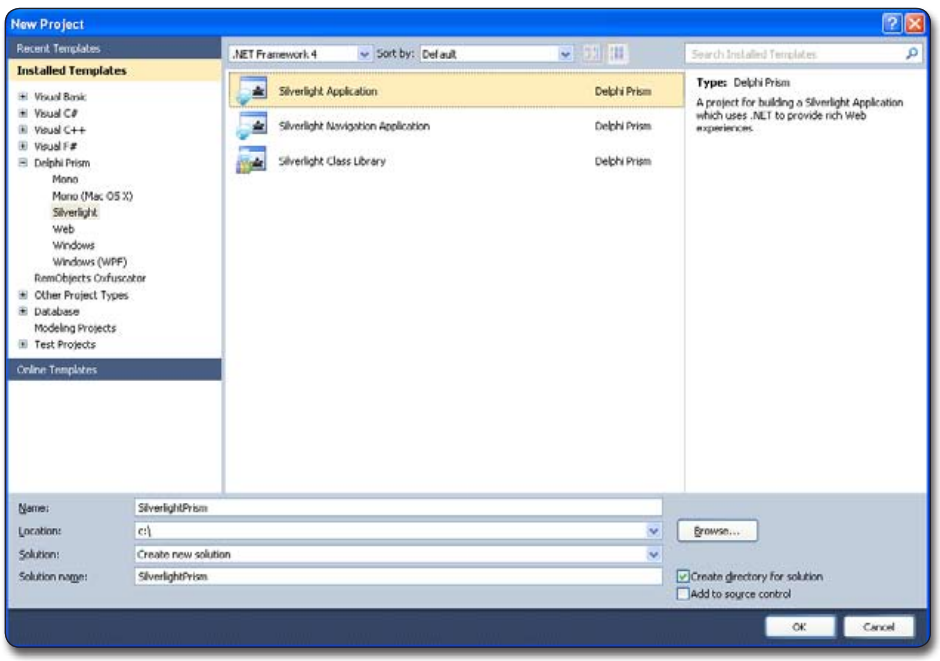


Figura 1. Criando um projeto Silverlight no Delphi Prism XE

Dê um duplo clique no botão e escreva no evento Click o código da **Listagem 2**.

Execute a aplicação e veja o resultado na **Figura 4.** Ao clicar no botão, temos a execução do evento. Mas veja aqui que, diferente da Web tradicional, não houve uma requisição ao servidor. Isso porque a aplicação já está toda carregada no ambiente de runtime do Silverlight no browser. Isso permite criar aplicações bem mais interativas e arrojadas, com experiência de usuário em um nível superior, mais rica (daí o termo "Rich" da sigla RIA).

Mais algumas coisas interessantes sobre o Silverlight que posso destacar aqui. Em sua versão 4, a tecnologia traz muitos novos recursos, principalmente para dar mais poder a execução de funcionalidades clientes. Agora é possível por exemplo ter acesso a WebCam a partir do plugin, ou ainda ao microfone. Também é possível ter acesso a algumas pastas do sistemas de arquivos do Windows, e até mesmo instanciar objetos COM, o que aumenta a possibilidade de funcionalidades que podemos incluir no lado cliente da aplicação. Lembre-se que ao desenvolver com tecnologias baseadas em HTML, o máximo que temos para enriquecer o cliente é JavaScript / AJAX.

Listagem 2. Código D#

```
method MainPage.button1_Click(sender: System.
Object; e: System.Windows.RoutedEventArgs);
begin
  TextBox1.Text := "Olá Mundo!";
end;
```

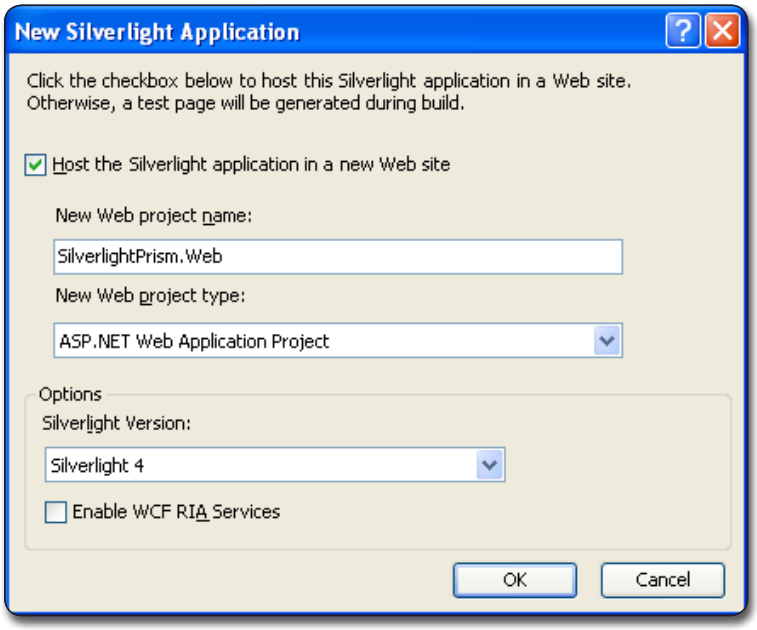


Figura 2. Opções para criação da aplicação Silverlight

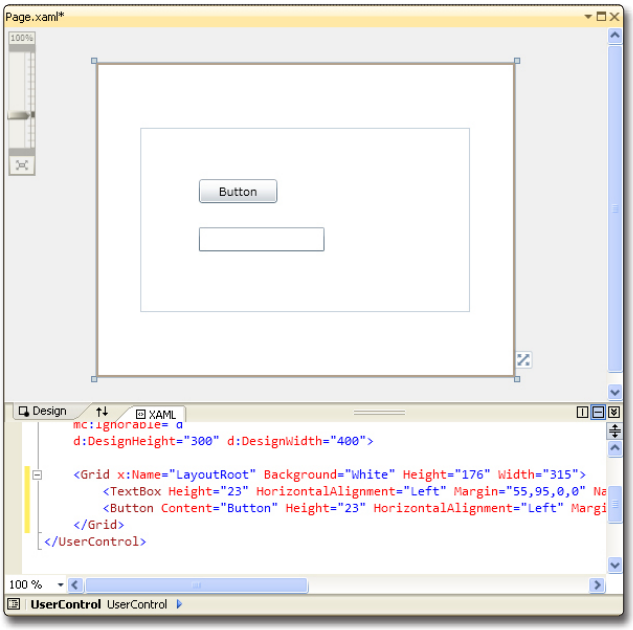


Figura 3. Desenhando uma aplicação Silverlight

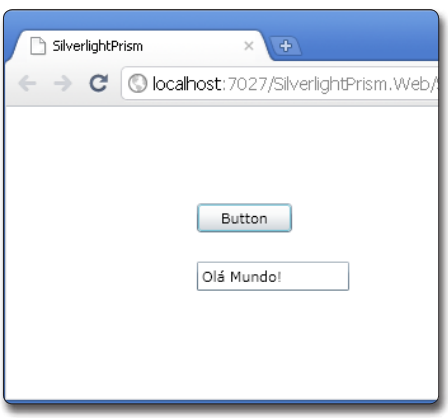


Figura 4. Aplicação Silverlight com Prism rodando no Google Chrome

Nota

Enquanto fechava este artigo, a Embarcadero acabara de anunciar a aquisição da KSDev, uma empresa que desenvolve bibliotecas VCL para WPF. Isso significa que, ao meu ver, teremos suporte à tecnologia WPF dentro do Delphi tradicional muito em breve, não se limitando apenas ao uso do Delphi Prism com Silverlight. Além disso, a Microsoft já havia anunciado que a versão beta do Silverlight 5 está prometida para o primeiro trimestre de 2011, que certamente também será suportado pelo Prism.

É claro, o acesso a todos esses recursos requer autorização do usuário. Uma outra grande novidade é a possibilidade de rodar as aplicações Out-Of-Browser (fora do browser). Basicamente o usuário pode dar um clique de direita na aplicação no browser e instalá-la por exemplo no Desktop local, como se fosse uma aplicação tradicional Delphi Win32. Veja que com isso o deploy fica extremente facilitado. No .NET temos uma solução semelhante já há algum tempo (desde o Visual Studio 2005), para soluções Windows Forms, chamada Click Once. Basicamente o usuário acessa uma URL e a aplicação é executada localmente no cliente. Quando uma nova versão da aplicação estiver disponível no servidor, ela automaticamente é atualizada. A grande novidade é que agora o WPF "Desktop" também desfruta do recurso.

Uma outra dica vai para os novos controles do Silverlight em sua versão 4, como um DataGrid incrementado. Outro controle muito interessante que tem sido usado pelos desenvolvedores é o PivotViewer, com a tecnologia Deep Zoom, para criar mosaicos com experiência extremamente inovadora para pesquisar dados. Basicamente pode-se criar grandes fotos a partir de várias outras menores. Coloquei alguns exemplos reais na sessão links que certamente vão despertar sua curiosidade.

Você pode estar se perguntando: mas como é feito o acesso a dados a partir desse tipo de aplicação, pois certamente não temos ADO.NET no lado cliente no browser. Aí que entra a arquitetura REST dos servidores WCF RIA Services, eles provêem uma interface para se obter e manipular dados obtidos de um servidor, usando XML ou JSON. Também é possível usar Web Services. Veja que aqui também entra em cena o novo DataSnap do Delphi Win32, que como agora suporta REST, pode alimentar dados para clientes Silverlight.

Outra dica ao construir aplicações Silverlight é o uso do padrão MVVM - Model View ViewModel, baseado no padrão MVC - Model View Controller. Fica aqui também a sugestão de estudo do Microsoft Expression Blend, uma ferramenta que pode ser usada em conjunto ao Visual Studio para criar interfaces Silverlight extremamente arrojadas. Um recurso interessante é a implementação de Behaviors, conjuntos de ações prontas para as mais diversas funcionalidades exigidas em interfaces WPF, como animações (a grosso modo, imagine como as poderosas ações prontas de um ActionList da VCL). Existe ainda o Silverlight Toolkit, um conjunto de controles para turbinar o desenvolvimento RIA. Não deixe também de conhecer mais sobre o mecanismo de DataBinding do WPF, onde podemos por exemplo ligar controles a praticamente qualquer coisa, até mesmo objetos, bem diferente da VCL onde podemos ligar controles Data-Aware somente a TFields. Este sem dúvida é um dos pontos fortes da tecnologia.

Guinther Pauli
guinther@devmedia.com.br
guinther.pauli@gmail.com
http://twitter.com/guintherpauli
http://gpauli.com

Desenvolve em Delphi há 15 anos, tendo utilizado a ferramenta desde a versão 1, passando pelo Turbo Pascal e Borland Pascal for Windows. Certificado em Delphi 3, 5, 6, 7, 2005, 2006, Delphi para Web e Delphi para Linux, palestrante em 5 edições da Borland Conference, beta tester de várias versões do Delphi, foi um dos pioneiros na divulgação da ferramenta no país. Certificado Microsoft MCP, MCAD, MCSD.NET, MCTS e MCPD (Visual Studio, C#, ASP.NET), utilizando a plataforma .NET desde a versão beta. Editor das Revistas ClubeDelphi, .NET Magazine e EASY .NET Magazine, professor de Pós-Graduação da UNIVEM-SP, na área .NET. Especialista em desenvolvimento Delphi, C#, .NET, ASP.NET, MVC, Visual Studio, atuando também em projetos de migração, orientando empresas e desenvolvedores na adoção de melhores práticas, ministra treinamentos em .NET focados em equipes Delphi. Foi arquiteto e desenvolvedor do site de doações on-line do projeto Criança Esperança da Rede Globo / Unesco.

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Microsoft Surface
http://www.youtube.com/watch?v=I4hkSSIFYII

Microsoft's Future Vision para o ano de 2019 (aqui tem muito WPF)
http://www.youtube.com/watch?v=g9JBSEBu2q8

Post interessante do Marco Cantu sobre o Silverlight
http://blog.marcocantu.com/blog/microsoft_silverlight.html

Especulações sobre o Silverlight
http://www.zdnet.com/blog/microsoft/microsoft-our-strategy-with-silverlight-has-shifted/7834
http://weblogs.asp.net/scottgu/archive/2010/11/04/silverlight-questions.aspx
http://community.devexpress.com/blogs/ctodx/archive/2010/10/29/a-story-for-halloween-is-silverlight-a-zombie.aspx

Projeto Moonlight
http://www.mono-project.com/Moonlight

Microsoft Silverlight
http://www.silverlight.net/
http://www.microsoft.com/silverlight/

Expression Blend
http://www.microsoft.com/expression/products/blend_overview.aspx

Silverlight 4 ToolKit
http://silverlight.codeplex.com/

Pivot da Copa do Mundo de 2010
http://pivot.metia.com/worldcup/

Pivot de Locais para Casamentos nos Estados Unidos
http://www.hitched.co.uk/wedding-venues/visual-search.htm

Hard Rock Deep Zoom
http://memorabilia.hardrock.com/

Parque Yosemite Deep Zoom (o zoom é tão profundo que é possível ver carros e pessoas)
http://www.xrez.com/yose_proj/Yose_result.html

Tigre
http://events.br.msn.com/verde/deep-zoom-tigre.aspx

Silverlight Toolkit em ação (live demo)
http://www.silverlight.net/content/samples/sl4/toolkitcontrolsamples/run/default.html

Conhecimento faz diferença!

Faça já sua assinatura digital ! | www.devmedia.com.br/es

Faça um *upgrade* em sua carreira

Em um mercado cada vez mais focado em qualidade, ter conhecimentos aprofundados sobre requisitos, metodologia, análises, testes, entre ouutros, pode ser a diferença entre conquistar ou não uma boa posição profissional. Sabendo disso a DevMedia lançou uma publicação totalmente especializada em Engenharia de Software. Todos os meses você pode encontrar artigos sobre Metodologias Ágeis; Metodologias tradicionais (document driven); ALM (application lifecycle); SOA (aplicações orientadas a serviços); Análise de sistemas; Modelagem; Métricas; Orientação à Objetos; UML; testes e muito mais. **Assine Já!**

DevMedia

Banco de Dados com

Utilizando threads para consultar dados de

Com o passar dos anos uma aplicação pode se tornar cada vez mais lenta, este processo faz parte de todo o ciclo de vida de um software e sempre fará parte da vida de suas aplicações, entretanto é correto afirmar que programas com acesso a bancos de dados tendem a ter uma durabilidade menor do que os outros, isto ocorre principalmente pelo preenchimento contínuo dos dados e da velocidade de acesso a eles. É claro que se pode contornar algumas situações criando índices ou até mesmo otimizando os resultados eliminando dados desnecessários em uma consulta, mas até que ponto essa aplicação suportará? Mesmo se migrar para outra linguagem de programação ou outro banco de dados se conseguirá alcançar resultados satisfatórios?

Com o intuito de liberar o usuário a continuar utilizando a aplicação ou até mesmo para executar processos demorados pode-se fazer uso das Threads (Tarefas) que nada mais são do que rotinas sendo executadas em conjunto com o programa sem que haja a necessidade de esperar que estes processos terminem. Isto garante que o software fique liberado plenamente para que o usuário possa continuar utilizando suas funções e jamais percebendo qualquer alteração de desempenho. Quando a aplicação é executada, ela é uma thread alocada em alguma parte da memória RAM e que no sistema operacional é conhecido como "processo". Ao observar o gerenciador de tarefas percebe-se que cada processo iniciado consome um valor x de processamento, logo, cada programa coloca-se em uma fila para que o sistema operacional possa processar seus dados e devolver as informações para continuar o processo. Um processo sempre vai possuir um PID (Process ID) e um Handle (número do processo na lista do sistema operacional).

Resumo DevMan

De que trata o artigo:

Iremos abordar conceitos sobre threads e criação de objetos em runtime, veremos os problemas mais comuns na implementação de threads e o que ocorre quando implementamos de forma errada uma consulta a uma base de dados. Ao final teremos pleno conhecimento para desenvolver códigos eficientes de threads manipulando dados do banco.

Para que serve:

Executar consultas ou manipular dados em segundo plano sem que o usuário tenha de esperar a tarefa ser concluída para continuar utilizando a aplicação. Elaborar novos conceitos de busca a grande quantidade de dados ou até mesmo enviar informações para um banco enquanto se está fazendo outra consulta, em outras palavras, realizar conexões e operações simultâneas ao banco de dados.

Em que situação o tema é útil:

Melhorar o desempenho de uma aplicação bem como melhorar sua usabilidade. Ao utilizar threads o processamento é melhor aproveitado.

Threads e banco de dados:

Neste artigo será abordado um conceito diferente de trabalhar com um banco de dados. É mostrado como permitir ao usuário continuar a utilizar o sistema enquanto uma tarefa em segundo plano se encarrega de efetuar grandes inserções ou atualizações na base. Também é possível, embora pouco prático, efetuar consultas e fazer com que os resultados sejam mostrados na tela sem que haja necessidade de aguardar com que o fetch de uma query termine, por exemplo.

Aplicações podem ter um ou mais processos (mais exatamente threads) tornando assim uma aplicação multitarefa. Pode-se ainda afirmar que cada thread instanciada dentro de uma aplicação ocupará uma parcela do tempo de CPU (tempo em que o CPU está processando alguma informação), o que garante que a aplicação poderá ser utilizada normalmente enquanto uma thread está sendo executada. Um dos exemplos mais práticos é a utilização de threads em jogos, onde existe um processo para o desenho das imagens, outro para o processamento de som e assim por diante. Outro exemplo muito bom é o dos programas especializados em baixar arquivos, como aceleradores de download ou programas de FTP que se utilizam de threads para permitir que o usuário continue aproveitando o programa para procurar mais conteúdo ou simplesmente para ver que algo está sendo feito e não que a aplicação tenha travado.

A implementação de uma thread não é deveras complicada, entretanto pela complexidade de entender como ela funciona muitos programadores optam por desistir de utilizá-la e passam a ignorar esta tecnologia que pode auxiliar muito em resolver problemas do cotidiano. De fato, depois de aprender a sua implementação, aumenta a vontade de utilizar muitas threads para automatizar as tarefas gerais que o programa deve desempenhar. O desafio

Threads
um banco sem comprometer sua aplicação

Marcos Aurélio Rocha

maior é entender e aprender, pois todo programador deve ao menos saber mais ou menos como fazer e, além disso, o tempo gasto no aprendizado é compensado pelo tempo ganho no momento da implementação quando há necessidade de executar uma rotina mais de uma vez de tempos em tempos.

Normalmente quando é necessário consultar, incluir, alterar ou excluir muitas informações de um banco de dados, também é preciso esperar que a rotina toda termine para que o usuário possa continuar a utilizar o programa. E se houvesse alguma forma de fazer isto enquanto o sistema continuasse destravado? E se fosse possível rodar folhas imensas de pagamento ou processar as férias de uma grande empresa enquanto o sistema está rodando sem que haja a necessidade de criar um programa separado para tal operação? Com um algoritmo corretamente estruturado, não haverá fronteiras para efetuar tão grandiosas operações e ainda permitirá que o sistema continue estável, o que valoriza muito o trabalho efetuado como um todo e garante com que rotinas pesadas possam ser processadas até mesmo enquanto se carrega a aplicação, por exemplo. Uma aplicação multitarefa pode e deve facilitar tanto a vida do usuário quanto a vida do programador, do contrário, se ela pender mais para um dos lados, sua implementação pode tornar-se um pesadelo.

Ao comparar a thread a algum dos componentes do Delphi, a Thread funciona praticamente como um Timer que de tempos em tempos pode executar um conjunto de rotinas para automatizar algo, entretanto é importante observar que o timer ainda depende tempo de processamento principal da aplicação para ser executado, ou seja, caso seja feita uma gigantesca consulta ao banco de dados dentro de um timer o usuário terá de esperar que todo o processo termine para poder continuar a utilizar o programa, e no caso da thread não. Ainda é importante salientar que uma thread não fica executando sozinha a menos que seja programado isso, do contrário, ela irá passar pelo seu método principal (o Execute ou Create dependendo da implementação do programador) e terminará o processo. O problema ocorre no fato de que threads são delicadas e merecem muita atenção, pois a qualquer desleixo, a aplicação pode e certamente irá travar. Tudo porque não se programou corretamente a consulta à base de dados de forma que o processo fica parado fazendo com que sua aplicação inteira pare de responder, resultado de uma falha de acesso aos dados por questões de sincronia.

Uma thread sempre irá executar as suas tarefas independentemente do aplicativo que a criou, portanto, se criar um método qualquer e solicitar sua execução, ela irá fazer isto em seu tempo de processamento separadamente, entretanto deve-se observar que métodos da thread que não manipulam variáveis privadas simples ou propriedades, devem ser executados com o método que faz a sincronia entre os processos da thread e da aplicação (processo principal).

A sincronia na thread é importante e deve ser sempre utilizada quando se manipula elementos que não pertencem a ela, como por exemplo, labels e edits pertencentes a um form ou Query (TQuery, TIBQuery etc.) mesmo que pertencentes a uma thread, tudo porque ao efetuar uma consulta a thread precisa esperar que o resultado venha, entretanto como a consulta é feita dentro da thread a aplicação continuará destravada. O nome do método utilizado para sincronizar dados entre as threads e o resto da aplicação é o Synchronize, onde o mesmo espera que seja passado como parâmetro a função que deseja sincronizar. A função (que irá sincronizar) deve ser do tipo **procedure** e não deve possuir parâmetros, portanto sempre que for preciso acessar variáveis ou objetos da thread ou até mesmo objetos fora dela é necessário sincronizar o método para evitar que a aplicação trave ocasionando erros como "CoInitialize was not called", erro ocasionado quando se tenta acessar um banco de dados com ADO ou "Canvas does not allow drawing" quando se tenta manipular algum objeto de um form, entre outros.

Nota

Ainda falando em threads e processos, pode-se também definir a afinidade de uma thread com o núcleo de um processador (1 para o primeiro núcleo e 4 para o último), ou seja, ter-se várias threads sendo executadas em diferentes núcleos do processador ao mesmo tempo. Isto permite jogar os processos mais pesados para o último núcleo da CPU. Ainda é possível também definir a prioridade de execução de uma thread, podendo fazê-la executar mais rapidamente ou mais lentamente.

Com relação aos objetos de bancos de dados não existem segredos, basta ter um pouco de conhecimento em orientação a objetos e será fácil gerenciar esta parte do problema. A única coisa que é preciso ter em mente é que cada processo que necessite conectar-se com o banco de dados deve ter seu próprio IBDatabase, ADOConnection, etc, assim como também devem ter sua própria transação, caso se aplique, e suas próprias Query, sendo assim, cada thread criada efetuará uma nova conexão completa com o banco de dados, portanto não se deve em hipótese alguma utilizar a conexão geral da aplicação, pois já é sabido o que pode acontecer se esta regra não for seguida à risca.

Listagem 1. Declarando e implementando uma thread

```
type
 TMinhaThread = class(TThread)
 public
  Contador: Integer;
  Stopped: boolean;
  procedure Execute; override;
 end;

implementation

procedure TMinhaThread.Execute;
begin
 inherited;
 while not Stopped do
 begin
  Inc(Contador);
  Sleep(100);
 end;
end;
```

Listagem 2. Executando e parando uma thread

```
procedure TForm1.BitBtn1Click(Sender:
TObject);
begin
 if MinhaThread = nil then
 begin
  MinhaThread := TMinhaThread.Create(True);
  MinhaThread.Contador := 0;
  MinhaThread.Resume;
 end;
end;

procedure TForm1.BitBtn2Click(Sender: TObject);
begin
 if MinhaThread <> nil then
 begin
  MinhaThread.Stopped := True;
  ShowMessage('O contador da Thread
  parou em:' +
   IntToStr(MinhaThread.Contador));
  FreeAndNil(MinhaThread);
 end;
end;
```

Implementando Threads

A implementação de uma thread não depende de unit separada, funções estranhas ou declaração de units na cláusula uses. Uma thread é uma classe padrão que se encontra na unit Classes.pas e é um objeto muito simples de ser implementado. Em suas propriedades, existe o acesso ao seu Handle, prioridade de execução e se ela já terminou sua execução. Seu método principal é o Execute, que deve ser obrigatoriamente implementado na declaração da classe. Por padrão, este método é executado apenas uma vez, embora seja possível implementar um loop para que ele fique executando até que seja enviada uma mensagem, por exemplo uma variável booleana, para que a thread possa terminar sua execução. Ao instanciar uma thread é possível definir se ela deve ou não ser instanciada em modo suspenso. No caso de ser True, ela irá aguardar um comando (Resume) para que prossiga com a execução, caso contrário ela automaticamente inicia suas tarefas designadas. A **Listagem 1** mostra uma classe que é um Thread.

Já a **Listagem 2** mostra como executar essa thread e como parar sua execução.

Neste simples exemplo pode-se ver como é fácil implementar e trabalhar com uma thread simples. Para visualizar como o processamento de uma thread afeta o sistema operacional, é possível abrir o Gerenciador de Tarefas do Windows na guia Desempenho antes de clicar no BitBtn1 para poder ter uma ideia do processamento em threads em uma aplicação e como o sistema operacional se comporta por trás da aplicação. É possível obter um nível de processamento muito maior comentando a linha que contém o comando Sleep(100), pois assim o processamento não aguardará 100 milissegundos.

Na criação da thread foi passado o valor True fazendo com que ela iniciasse inicialmente parada para que se iniciasse o valor da variável Contador em 0. É claro que na prática este exemplo não se aplica perfeitamente bem, pois quaisquer variáveis ou até mesmo objetos dentro de uma thread podem e devem ser inicializados no construtor Create, método este que pode ser sobrescrito normalmente na declaração da classe desde que seja obedecido o método antecessor, que no caso é o próprio Create da TThread. O código a seguir mostra o que deve ser adicionado na classe TMinhaThread para sobrescrever o construtor e já seja setado o valor da variável Contador:

```
constructor Create(CreateSuspended: Boolean);

constructor TMinhaThread.Create(CreateSuspended:
Boolean);
begin
 Stopped := False;
 Contador := 0;
end;
```

A importância da sincronia

Apesar de falar que as threads são independentes da aplicação, quando se precisa que elas interajam entre si é necessário utilizar-se do método Synchronize, que garante a sincronia entre as diferentes threads atuando no programa naquele momento. De fato, se não for feito o uso desta rotina fatalmente ocorrerão travamentos inexplicáveis e praticamente impossíveis de serem detectados. Um exemplo disso é a tentativa de efetuar uma busca ao banco de dados simplesmente utilizando a query e/ou a conexão principal da aplicação.

Ao tentar efetuar uma consulta com o banco de dados a thread pode ocasionalmente funcionar perfeitamente, mas após qualquer implementação em outra thread, ou mesmo processamento de dados ao mesmo tempo em que ela está em execução faz com que o programa trave e simplesmente deixe de responder definitivamente. Tudo porque ocorre internamente uma quebra de sincronia entre os processos resultante em erro principalmente na biblioteca netbios.dll nativa do Windows, porém tal erro nunca chega aos olhos do usuário e dificilmente o programador se dá conta de que a implementação dos métodos da thread não está em sincronia com o resto da aplicação, por isso é utilizado o Synchronize. Dessa forma não há qualquer risco de a aplicação travar quando a thread estiver atualizando o Memo1 e outros processos também estiverem manipulando os componentes de Form1. A **Listagem 3** mostra um exemplo desse tipo de operação.

Aqui duas threads do mesmo tipo compartilham o mesmo componente visual que faz a interface com o usuário, no caso um Memo, mostrando nele o valor atual contido em sua variável chamada Contador em um intervalo de meio segundo cada uma. É fato que ambas irão preencher o Memo1 de acordo com a ordem de criação, sendo respectivamente a MinhaThread1 e a MinhaThread2. O resultado mostra que ambas as threads estão em sincronismo com a aplicação, o que não quer dizer necessariamente que ambas as threads estejam aguardando cada uma ser executada. A questão de sincronia diz que cada thread deve, naquela hora, aguardar que a aplicação possa processar os comandos das outras para que então possa executar novas tarefas. Ao final o resultado não é afetado na questão de desempenho, pelo contrário, se bem implementadas as threads podem oferecer ganhos significativos em processamentos permitindo flexibilidade em suas aplicações, pois o tempo de espera para se processar a mensagem é significativamente minúsculo.

Consultando o banco sem Synchronize

Como visto anteriormente, o método Synchronize é utilizado para que toda a aplicação trabalhe em conjunto evitando travamentos decorrentes do excesso de processamento requisitado aos componentes do sistema operacional. Mesmo utilizando conexões e query própria, a thread não fica em sincronia com o resto da aplicação se necessitar manipular componentes visuais, entretanto se a thread utilizar sua própria conexão para incluir, alterar ou excluir dados sem que qualquer retorno seja enviado ao usuário, o método Synchronize pode ser dispensado.

A esta altura do desenvolvimento vem a pergunta: Então pra que utilizar o Synchronize se eu posso manipular o banco de dados internamente na thread? Ora, é interessante não somente para o usuário, mas para você programador saber que determinada tarefa está sendo executada em background no caso de eventuais mensagens de erro. Na prática muitas rotinas executadas dentro de uma thread não necessitam de sincronia com o resto da aplicação, embora seja preciso estar atento para definir se um método exige ou não este método. É possível, por exemplo, criar uma aplicação com várias threads manipulando os componentes de um Form, porém corre-se o sério risco de acabar manipulando um componente que outra thread já possa estar utilizando e isto não deve ser feito sem o Synchronize.

O exemplo da **Listagem 4** faz uma consulta ao banco de dados Interbase a partir de uma Query contida em uma thread. Neste caso não será utilizado qualquer método que se comunique com o resto da aplicação fazendo apenas uma consulta e algumas alterações no banco de dados. Tudo é executado diretamente na procedure Execute da thread onde a mesma automaticamente é eliminada da memória graças a propriedade FreeOnTerminate sendo definida como True. Esta propriedade diz a thread que libere sua instância da memória assim que terminar, ou seja, após passar pela última linha de seu método Execute. No caso de um método recursivo, a thread irá ser liberada da memória única e exclusivamente quando sua primeira chamada chegar ao fim. Foi utilizado um Timer com a propriedade Interval setada em 1800000 (30 minutos) onde o mesmo cria e executa a thread que após seu término libera seus componentes da memória automaticamente.

Consultando o banco de dados com Synchronize

A implementação do método Synchronize com o banco de dados enquadra-se perfeitamente no exemplo aplicado anteriormente onde é preenchida a numeração de 1 a 100 em um componente Memo em um Form. Como anteriormente, será criado um método sem parâmetros que se encarregará de efetuar as atualizações em tela ou, caso se aplique, a outras threads correntes e assim por diante. Enquanto estiver dentro de um método invocado por Synchronize, pode-se acessar as variáveis e objetos da thread sem problemas, afinal é ela que está sincronizando as informações com o resto do sistema, portanto é perfeitamente normal utilizarmos variáveis e objetos pertencentes a ela.

Listagem 3. Utilizando método Synchronize para preencher um Memo

```
type
 TMinhaThread = class(TThread)
 private
  Contador: Integer;
  procedure AtualizaMemo;
 public
  procedure Execute; override;
 end;
...
procedure TMinhaThread.Execute;
begin
 inherited;
 Contador := 0;
 while Contador < 100 do
 begin
  Synchronize(AtualizaMemo);
  Inc(Contador);
 end;
end;

procedure TMinhaThread.AtualizaMemo;
begin
 Form1.Memo1.Lines.Add(IntToStr(Contador));
end;
```

No exemplo a seguir, da **Listagem 5** pouca coisa do código anterior é alterada, afinal tudo já está pronto para efetuar a consulta e a alteração do banco de dados, apenas se executa o FetchAll da Query para poder ter a quantidade total de registros trazidos no select para gerar as informações visíveis ao usuário. E como o exemplo foi feito para fins de explicação, o SQL pode ser adaptado para poder testar com algum banco de dados de exemplo e verificar o resultado final da thread. Também é impossível utilizar uma procedure com parâmetros, por exemplo, MinhaProcedure(const S: String) no método Synchronize portanto, é preciso atentar aos erros emitidos pela IDE durante a compilação. Eles ajudam a identificar problemas de implementação e poupam vários minutos tentando descobrir de onde vem o erro de compilação.

Na prática, as informações do progresso da atualização poderiam ser mostradas em um ProgressBar posto em um StatusBar ou apenas em um Label. O fato é que com este pensamento pode-se manipular outras threads, objetos em outro Form, arquivos etc. O programador deve ter em mente que pode ter quantas procedures de sincronização quanto achar necessário, desde que elas sejam invocadas com o Synchronize.

Listagem 4. Consultando e manipulando o banco de dados com threads

```
type
 TMinhaThread = class(TThread)
 private
  FDatabase: TIBDatabase;
  FTransaction: TIBTransaction;
  FQrConsulta: TIBQuery;
  FQrDados: TIBQuery;
 public
  constructor Create(CreateSuspended: Boolean);
  destructor Destroy; override;
  procedure Execute; override;
 end;

constructor TMinhaThread.Create(CreateSuspended: Boolean);
begin
 inherited Create(CreateSuspended);
 FDatabase := TIBDatabase.Create(nil);
 FTransaction := TIBTransaction.Create(FDatabase);
 FQrConsulta := TIBQuery.Create(FDatabase);
 FQrDados := TIBQuery.Create(FDatabase);
 FDatabase.DatabaseName := DataModule1.Database.DatabaseName;
 FDatabase.Params := DataModule1.Database.Params;
 FDatabase.AllowStreamedConnected := False;
 FDatabase.LoginPrompt := false;
 FDatabase.DefaultTransaction := FTransaction;
 FTransaction.DefaultDatabase := FDatabase;
 FQrConsulta.Database := FDatabase;
 FQrConsulta.Transaction := FTransaction;
 FQrDados.Database := FDatabase;
 FQrDados.Transaction := FTransaction;
end;

destructor TMinhaThread.Destroy;
begin
 FreeAndNil(FQrDados);
 FreeAndNil(FQrConsulta);
 FreeAndNil(FTransaction);
 FreeAndNil(FDatabase);
 inherited;
end;

procedure TMinhaThread.Execute;
begin
 inherited;
 try
  FDatabase.Connected := True;

  if not FTransaction.InTransaction then
   FTransaction.StartTransaction;

  FQrConsulta.SQL.Clear;
  FQrConsulta.SQL.Add('Select cli_codigo, cli_ultimavenda from Cliente');
  FQrConsulta.SQL.Add('Where cli_situacao = ''ATIVO''');
  FQrConsulta.Open;
  while not FQrConsulta.Eof do
  begin
   { Clientes com inatividade há mais de 90 dias }
   if FQrConsulta.FieldByName('cli_ultimavenda').AsDateTime + 90 < Date then
   begin
    FQrDados.SQL.Clear;
    FQrDados.SQL.Add('Update Cliente set cli_situacao = ''INATIVO''');
    FQrDados.SQL.Add('Where cli_codigo = '+FQrConsulta.FieldByName
    ('cli_codigo').AsString);
    FQrDados.ExecSQL;
   end;
   FQrConsulta.Next;
  end;
  FTransaction.CommitRetaining;
 except
  on E: Exception do
  begin
   FTransaction.RollbackRetaining;
  end;
 end;
end;

procedure TForm1.Timer1Timer(Sender: TObject);
var
 MinhaThread: TMinhaThread;
begin
 MinhaThread := TMinhaThread.Create(True);
 MinhaThread.FreeOnTerminate := True;
 MinhaThread.Resume;
end;
```

Listagem 5. Implementando consultas com Synchronize

```
type
 TMinhaThread = class(TThread)
 private
  ...
  procedure MostraProgresso;
 public
  ...
 end;

procedure TMinhaThread.MostraProgresso;
begin
 Form1.Label1.Caption := 'Registro: '+IntToStr(FQrConsulta.RecNo)+
 ' de '+IntToStr(FQrConsulta.RecordCount);
 Form1.ProgressBar1.Max := FQrConsulta.RecordCount;
 Form1.ProgressBar1.Position := FQrConsulta.RecNo;
 Application.ProcessMessages;
end;

procedure TMinhaThread.Execute;
begin
 inherited;
 ...
  FQrConsulta.Open;
  FQrConsulta.FetchAll;
  while not FQrConsulta.Eof do
  begin
   { Clientes com inatividade há mais de 90 dias }
   if FQrConsulta.FieldByName('cli_ultimavenda').AsDateTime + 90 < Date then
   begin
    FQrDados.SQL.Clear;
    FQrDados.SQL.Add('Update Cliente set cli_situacao = ''INATIVO''');
    FQrDados.SQL.Add('Where cli_codigo = '+FQrConsulta.FieldByName
    ('cli_codigo').AsString);
    FQrDados.ExecSQL;
   end;
   FQrConsulta.Next;
   Synchronize(MostraProgresso);
  end;
 ...
end;
```

Conclusão

É concluído claramente então que o método Synchronize se faz necessário para interação com outros processos da aplicação. O método é comumente utilizado em aplicações de grande porte no quesito automação e praticidade. Os horizontes se abrem muito mais quando se faz uso desta técnica que sempre esteve ali, mas poucos de nós fomos capazes de enfrentar o medo do desconhecido e passar a utilizá-la. De fato, pode-se ver vários programas que utilizam threads e com certeza o método Synchronize, entre eles os Navegadores de internet que utilizam abas, desfragmentadores de arquivos, antivírus e muitos outros.

O fato de se utilizar um banco de dados também não deve ser o impasse para a não utilização das threads. Aqui foi envolvido um conceito de uso de banco de dados para visualizar o quão é importante deixar em segundo plano, ou seja, deixar rodando em background rotinas pesadas onde o usuário teria de ficar esperando quando o programador poderia simplesmente fazer com que estas rotinas fossem executadas liberando a utilização do software. Realmente a praticidade da metodologia leva à excelência o trabalho da equipe, basta apenas dar asas à imaginação e automatizar suas consultas e rotinas de dados com o uso das threads não se esquecendo da sincronia.

Marcos Aurélio Rocha
marcosrocha85@gmail.com
Bacharel em Sistemas de Informação atuando há mais de 7 anos com desenvolvimento de aplicações em Delphi com banco de dados Firebird e SQLServer e Web com PHP, Ajax e MySQL.

Threading in Delphi
http://delphi.about.com/od/kbthread/Threading_in_Delphi.htm

Synchronizing Threads and GUI in Delphi application
http://delphi.about.com/od/kbthread/a/thread-gui.htm

Delphi threading by example
http://edn.embarcadero.com/article/22411

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Cursos Online

WebMobile PLUS

A Revista **WebMobile** oferece para seus assinantes uma série de Cursos Online de alto padrão de qualidade .

Conheça abaixo os cursos já disponíveis:
www.devmedia.com.br/curso/webmobile

Curso em .NET: **Aplicação completa de orçamento Doméstico no Visual Studio 2005**

Curso em Java: **Introdução ao uso de Web Services em Java ME**

A sua melhor opção de aprendizagem!
Assine a **WebMobile** e
Comece já seu treinamento!
www.devmedia.com.br/assine

DevMedia Mais Informações: www.devmedia.com.br/central - Tel.: 21 3382-5038

Delphi Prism

Melhorias da linguagem – Parte 1

O Delphi Prism inclui uma nova estrutura de repetição adicionada à linguagem. Esta estrutura é representada por um loop que apresenta uma construção de forma bastante simples e mantém a repetição indefinidamente, somente parando quando for quebrada por uma instrução que condicione sua saída, como por exemplo, uma instrução break.

A **Listagem 1** ilustra um exemplo utilizando este formato de repetição. No código a estrutura irá repetir mensagens na tela até que a condição determinada pela estrutura condicional if seja satisfeita, finalizando as repetições. Se a variável x for definida, inicialmente, com valor zero, serão apresentadas onze mensagens na tela, cada uma indicando o seu numeral correspondente ao incremento da variável x. Além da instrução break, a instrução continue também pode ser utilizada.

A **Listagem 2** apresenta uma variação do código, mas com o mesmo objetivo final, obtendo-se os mesmos resultados (ilustrado apenas para fins didáticos).

Case

A estrutura condicional case fornece uma forma simples para processar ou selecionar diferentes valores contidos em uma lista de possíveis valores e determinar o que deverá ser executado para cada um desses valores. Neste aspecto, o Delphi Prism tem melhorias comparadas ao Delphi Win32 tradicional, utilizando esta estrutura condicional, também apresenta suporte para o uso de strings e objetos em instruções case.

A **Listagem 3** demonstra o uso de case com strings. A variável de nome tipo e a variável mensagem são do tipo string e, de acordo com cada estrutura condicional

Resumo DevMan

De que trata o artigo:

O artigo demonstra algumas das principais melhorias e palavras reservadas da linguagem do Delphi Prism. Neste artigo, abordaremos preferencialmente recursos que possuem significativas melhoras / incrementos, ou mesmo que não existem, se comparados aos recursos da linguagem Delphi Win32 tradicional.

Para que serve:

As melhorias na linguagem servem para tornar a codificação mais enxuta e simples, ao mesmo tempo em que adicionam recursos poderosos presentes em linguagens orientadas a objetos mais modernas.

Em que situação o tema é útil:

O tema pode ser utilizado para ampliar os conhecimentos e esclarecer dúvidas do que está presente no Delphi Prism, e que é considerado melhoria ou novidade na linguagem, assim, possibilitando uma melhor utilização dos recursos presentes acelerando e aperfeiçoando o desenvolvimento de sistemas gerando um ganho de produtividade e qualidade no que está sendo criado.

Delphi Prism – A linguagem:

O Delphi Prism, lançado pela Embarcadero, trouxe o desenvolvimento da linguagem Object Pascal para o .Net Framework a qual permite o uso do compilador Oxygene da Remote Objects em conjunto com o IDE do Visual Studio. Recentemente com a chegada do Delphi XE, a Embarcadero lançou também a nova versão do Delphi Prism que está cheia de novidades. Neste artigo serão demonstradas as novidades que dizem respeito à linguagem, focando principalmente nos incrementos, ou seja, o que tem no Prism que não tem no compilador Win32. Serão apresentados alguns conceitos como loop, a estrutura case com suporte aperfeiçoado possibilitando o uso de Strings e objetos, o uso de locking permitindo a construção de trechos de código críticos principalmente para uso em ambientes multitarefas etc. Também será visto o conceito de invariants para definição de estados fixos aos objetos, diferente do que ocorre com condições pré e pós. O conceito de future utilizando o uso de funcionalidades em paralelo, dessa forma, ampliando o desempenho de um aplicativo. Dando sequência ao paralelismo aparecem for e for each como estruturas de repetição condicionadas a variáveis ou a coleções de itens, além do mais, o uso de parallel relacionado a for e a for each possibilitando o seu paralelismo nessas estruturas. Após, é citado Nullable permitindo inicializar variáveis como nulas, nos casos em que elas devam ser iniciadas com algum valor. Sequence aparecendo como um tipo especial na linguagem que irá representar uma coleção de elementos de determinado tipo. O comando where como uma opção de filtragem de resultados garantindo assim resultados mais precisos do que se está procurando ou querendo indicar. E, por fim, os tipos nested que indicam a definição de um tipo utilizando um membro de outro tipo previamente definido.

Listagem 1. Código para demonstrar o uso da instrução de repetição loop

```
loop
begin
 inc(x);
 MessageBox.Show(x.ToString());
 if x > 10 then
  break;
end;
```

Listagem 2. Código adaptado para demonstrar o uso da instrução de repetição loop

```
loop
begin
 inc(x);
 MessageBox.Show(x.ToString());
 if x > 10 then
  break
 else
  continue;
end;
```

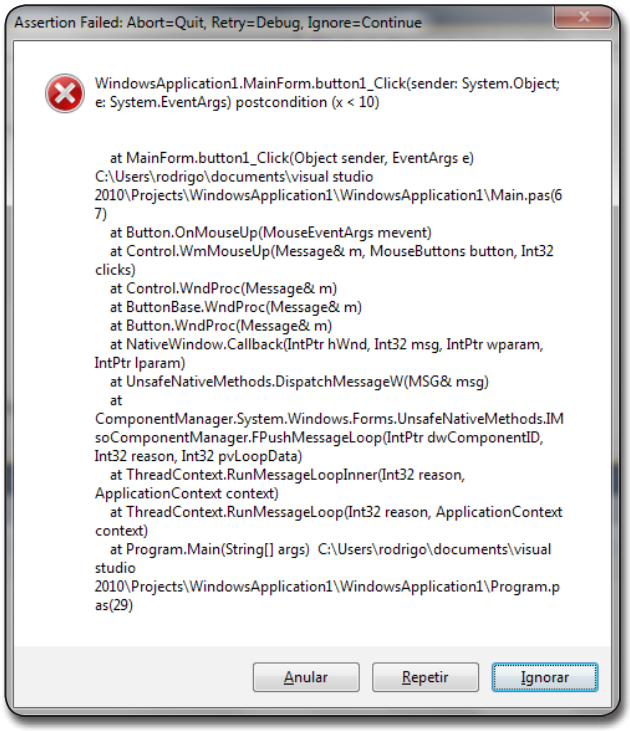


Figura 1. Mensagem indicada na tela ao validar um código através de uma condição

indicada pelos diferentes tipos: pessoal, profissional, residencial ou outros não declarados, será informada uma mensagem pré-definida na variável utilizada.

Como uma versão adicional e expandida do padrão case, existem as declarações case type que permitem executar diferentes resultados utilizando por base o tipo da variável informada. Na **Listagem 4** é possível visualizar um exemplo de código utilizando case type. O código indica que a instrução case type utilizará informações relativas ao tipo da variável informada, neste caso, button1, e dessa forma o seu resultado será um Button que será apresentado na mensagem descrita.

Ensure

Representa uma pós-condição que será avaliada e que se faz necessária ser verdadeira para dar continuidade ao fluxo de execução de um código. Ensure é utilizada, principalmente, ao final de um método tendo por principais finalidades a verificação da validade de algum parâmetro de entrada, variáveis, resultados ou para determinados estados de objetos exigidos pelo método onde está sendo utilizado.

A **Listagem 5** demonstra a utilização de ensure em um determinado trecho de código. Após a execução da estrutura de repetição loop, ao final, é indicado o código que irá representar o ensure, este código irá realizar uma validação na variável x. Neste caso, a condição não será satisfeita, pois a variável x apresenta um valor superior a 10. Dessa forma, ensure irá gerar uma exceção de código e uma mensagem será exibida na tela, a **Figura 1** demonstra a mensagem que será indicada. É possível verificar a pós-condição e a indicação do que era analisado, neste caso, x<10 e a informação das linhas onde a condição não foi validada e em qual o arquivo que está essa condição.

Locking

A palavra reservada locking possibilita que seja construído um bloco de código crítico, ou seja, thread-safe, permitindo que somente uma linha de execução utilize os recursos alocados no referido bloco de código, que somente serão liberados após uma thread executar o código, uma de cada vez. A sua utilização é comumente com a programação multi-thread, para resolver de forma simplificada, o que antigamente era descrito utilizando funcionalidades do próprio sistema operacional através de mutex e seções críticas.

A **Listagem 6** exemplifica, de forma simples, o uso de locking para um objeto. No código o uso de locking garante que naquele momento a utilização de treeview1 somente será modificada por este trecho de código, assim, durante a remoção de um de seus nós da treeview, somente este código que poderá acessá-lo, representando de forma consistente e única a alteração.

Listagem 3. Código utilizando strings em uma estrutura case

```
case tipo of
 'pessoal': mensagem:='Telefone pessoal';
 'profissional': mensagem:='Telefone profissional';
 'residencial': mensagem:='Telefone residencial';
 else
   mensagem:='sem tipo definido';
end;
```

Listagem 4. Exemplo de utilização de uma estrutura case type

```
case button1 type of
 System.Windows.Forms.Button : MessageBox.Show('é um botão');
 System.Windows.Forms.Label : MessageBox.Show('é um label');
end;
```

Listagem 5. O uso de ensure em um método

```
x := 0;
loop
begin
 inc(x);
 MessageBox.Show(x.ToString());
 if x > 10 then
   break
end;
ensure
 x < 10;
```

Listagem 6. O uso de locking

```
locking TreeView1 do
begin
 treeView1.Nodes.RemoveAt(1);
end;
```

Listagem 7. A utilização de invariants

```
type
 TPessoa = class
 private
   FIdade: Integer;
 private invariants
   FIdade < 120;
 public
   constructor Create;
   property NomePessoa: string;
   property Idade: Integer read FIdade write FIdade;
 end;
```

Listagem 8. A utilização de future em uma função de cálculo

```
function ThreeNode.Sum: Int32;
begin
 var l: future Int32 := async Left.Sum;
 var r: Int32 := Right.Sum;
 result := r + l;
end;
```

Invariants

Diferentemente do que ocorre com as condições pré e pós-utilizadas para dar continuidade ao fluxo de código, um invariant é utilizado para definir um estado fixo que os objetos deverão respeitar a qualquer momento. Nesse aspecto, os dados serão informados e provavelmente armazenados, dessa forma em alguns casos poderá ocorrer a necessidade de validar e condicionar limitações específicas para os dados. Assim, invariant possibilita criar restrições que não possam ser violadas em qualquer momento da execução do código. Invariant poderá ser público ou privado, dependendo de quantas vezes o seu conteúdo será verificado. De forma pública a verificação ocorrerá ao final de cada método público, e se ocorrer alguma falha, uma exceção será gerada. O invariant privado será verificado ao final de cada chamada de método, seja público ou privado.

Na **Listagem 7** está presente uma demonstração de código utilizando invariant. O código indica a definição de uma classe TPessoa. Nesta classe existem os atributos NomePessoa e Idade, e além disso a definição de um invariant privado relacionado a idade da pessoa que deverá ser inferior a cento e vinte anos. Esta definição será monitorada no fluxo de execução do código, a cada definição ou atribuição da propriedade em questão, no caso o atributo idade, se a condição não for respeitada, uma exceção será gerada no aplicativo.

Futures

Futures representam um novo conceito para facilitar o desenvolvimento em computação paralela, assim utilizando recursos de sistemas multiprocessados. Apresentam variáveis fortemente tipadas e seus valores são tratados de forma distinta onde podem ser calculados ou ainda não estão em fase de cálculo, mas possuem garantia de que a operação será realizada quando necessário. O Future trouxe novas funcionalidades que permitem a utilização de tarefas em paralelo em um aplicativo, ou seja, a capacidade de executar múltiplas threads ao mesmo tempo e simultaneamente. Neste aspecto os aplicativos poderão utilizar vários processadores sem a preocupação de trabalhar diretamente com o processamento em paralelo, logo facilitando a programação com as suas bibliotecas de computação paralela. Dessa forma, permitindo o uso de recursos dos computadores mais atuais, onde praticamente todos são multiprocessados.

A **Listagem 8** demonstra a utilização de future em uma função de cálculo. Este código define o cálculo da parte esquerda e direita de uma árvore através de seus nós. Primeiramente ocorre o cálculo do lado esquerdo e depois o lado direito, por final, o somatório representa o cálculo final como resultado. Observe que o cálculo dos nós do lado esquerdo poderá demorar um tempo considerável, e o cálculo do lado direito

independe do lado esquerdo, somente na última linha onde será indicado o somatório dos dois lados, dessa forma, torna-se um cálculo ideal para a utilização de future.

A declaração de future no código anterior possibilita dizer que em algum momento a variável l receberá o cálculo Left.Sum, e este momento poderá demorar alguns instantes. Além disso, a declaração assync possibilita reservar um segmento separado de tarefa para o cálculo da variável l, dessa forma, realizando de maneira assíncrona este cálculo, independentemente do cálculo realizado com a variável r.

Utilizando este modelo de programação com future vale destacar que o sistema poderá apresentar três estados diferentes para a realização do cálculo da variável l, ao chegar no final do fluxo de execução, na última linha, onde acontecerá a soma de valores. O primeiro estado poderá ser o resultado do cálculo, onde já foram processados todos os valores e realizadas as atribuições necessárias à variável l, possuindo o valor pronto para ser utilizado no somatório final. O segundo estado, definido quando chegar ao final o sistema, ainda está realizando o cálculo e o valor ainda não está atribuído, ou seja, o sistema está calculando. O terceiro estado indica a necessidade de realizar o cálculo, assim o sistema ainda nem iniciou a realização da tarefa, provavelmente por motivos de processamento, como por exemplo, os núcleos restantes estarem ocupados com outras tarefas.

Ao utilizar future um dos três estados será representado, neste caso, pela variável l, e o aplicativo deverá tratar de forma adequada cada uma das situações. No primeiro caso, simplesmente irá utilizar o valor já calculado para o somatório, no segundo, será bloqueado e ficará aguardando o cálculo dos valores, até que o valor final seja determinado, e o terceiro e último caso, será cancelada alguma tarefa para dar prioridade ao cálculo solicitado evitando assim segmentações desnecessárias, permitindo em seguida realizar o somatório final.

Existem dois modelos para utilização de future, um modelo assíncrono e um modelo síncrono. O modelo assíncrono é o que foi demonstrado anteriormente, já em um modelo síncrono as tarefas não serão realizadas em um segmento separado de trabalho, mas será avaliado e realizado como parte do segmento de chamada na primeira vez que for acessada. Analisando do ponto de vista funcional, os dois modelos apresentam o mesmo resultado e sua única diferença é na forma como o cálculo é realizado, ocorrendo no mesmo segmento ou em local distinto.

A **Listagem 9** descreve um cenário típico para um modelo síncrono onde ocorre a criação efetiva de um objeto. A classe compras possui um atributo fValores que é utilizado em conjunto com future e relacionado a Produtos, onde cada produto terá o seu valor. Assim, em fValores estará presente o total de valores relativos às compras.

Listagem 9. Definição de um modelo síncrono

```
type
  Compras = public class
  private
    fValores: future Produtos := new Produtos();
  end;
```

Listagem 10. Código para uma estrutura de repetição for

```
for i: Integer := 0 to 10 step 2 do
begin
...
end;
```

Listagem 11. Exemplo de utilização de for each

```
for each valor in minhalista do
```

Listagem 12. Representação do uso de nullable

```
var i: nullable Integer;
```

Listagem 13. Código para exemplificar a utilização de sequence

```
var nomes: sequence of String;
nomes := ['Rodrigo', 'Devmedia'];
nomes := new List<String>;
nomes := getNomes();
```

For

Dentre as estruturas de repetição, encontra-se o for, que é uma instrução utilizada em Delphi Prism para realizar uma iteração por um determinado número de vezes utilizando uma variável pré-definida ou algum outro tipo de estrutura. Seu princípio de funcionamento é baseado em um incremento ou decremento, dependendo da ordem a ser utilizada e sua contagem será até um determinado limite previamente definido. Semelhante ao for do Delphi Win32, porém bem mais incrementado, a começar pela declaração do tipo que pode ser feita in-line.

O código da **Listagem 10** demonstra a utilização de uma estrutura de repetição simples. No exemplo a estrutura de repetição for é utilizada referenciando a variável i, e esta variável será incrementada até atingir o limite definido (valor 10). Caso existisse a definição de um decremento, seria utilizada a palavra downto ao invés de to e a contagem seria ao inverso, começando em dez até chegar ao valor zero, semelhante ao exemplo mencionado, mas em ordem inversa.

Uma variação para estruturas de repetição é a utilização de for each que pode ser utilizada para interagir sobre uma estrutura que referencie alguma interface semelhante a IEnumerable. A sua execução é realizada a partir do tipo que é inferido na utilização de IEnumerable, onde será gerada uma falha caso não seja possível utilizar o tipo, e a utilização do laço de repetição levará em conta somente os valores que realmente são do tipo mencionado, neste caso, o seu final irá ocorrer quando existir a referência a um valor nulo, representando que não existem mais itens a serem utilizados. O exemplo da **Listagem 11** indica esta situação mencionada.

Nullable

O Delphi Prism introduziu o suporte a tipos nulos nativos na linguagem. A sua utilização apresenta o objetivo de permitir atribuir valores nulos a variáveis.

Dessa forma, possibilitando inicializar variáveis que muitas vezes contêm elementos que não podem ser adicionados sem que seja atribuído algum valor. Na **Listagem 12** está representado o uso de um nullable. Quando um tipo nullable é definido, sua representação utilizará um System.Nullable de chamada que permitirá acesso a um valor nulo na variável representada, tornando o uso de tipos nulos bastante transparente e exigindo menos código para o seu trabalho, possibilitando também a inserção de valores nulos quando em uso com tabelas de um BD.

Sequence

Sequences são definidas como um tipo especial utilizado na linguagem e podem ser analisadas como uma coleção de elementos de determinado tipo semelhantes a um array. Não implicam em uma forma específica de armazenamento de dados, podendo representar coleções de elementos disponíveis em uma ordem específica. Sequence permite utilizar armazenamento de vários tipos de dados, listas, árvores binárias e desenvolvimento de implementações personalizadas.

O trecho de código da **Listagem 13** mostra a utilização de Sequence em Delphi Prism. No exemplo ocorre a definição de uma variável chamada nomes como sendo uma sequência de strings. Sequence representa um conceito de tipo abstrato indicando como os dados serão acessados, mas sem definir a forma de como os dados serão armazenados, assim, permitindo instanciar um tipo específico que defina o comportamento dos dados a serem utilizados. Como pode ser visto no código anterior, uma vez definida a variável utilizando Sequence, ela poderá ser usada de diversas maneiras, independentemente da forma como os dados serão armazenados, introduzidos ou gerados.

Pela definição é possível analisar e verificar que Sequence define uma forma de como acessar os dados, mas está totalmente livre da maneira de como esses dados irão existir. Ao utilizar a variável e indicar um array com nomes, pode-se observar que Sequence permite vários formatos de utilização da variável para a passagem de valores. No primeiro caso, um array representa Sequence, no segundo, a definição e utilização de uma lista, e no terceiro e último caso, a chamada de uma função que irá retornar no formato compatível com Sequence.

Caso Sequence realize referência a um array ou uma lista de objetos, os dados serão acessados diretamente em memória, onde buscará as referências existentes na lista ou array, no entanto, se for uma implementação personalizada ou que realize determinadas iterações, por exemplo, estruturas de repetição, os dados poderão ser obtidos de fontes externas ou gerados no momento de acesso.

Sequence permite utilizar os elementos individualmente de forma tardia, até o momento que eles sejam referenciados ou utilizados. Como exemplo, pode-se ter o acesso a informações contidas em um banco de dados, este banco poderá conter centenas de registros presentes nas tabelas, e esses registros serão transferidos para a memória local somente quando as linhas individuais onde ocorre o uso de Sequence sejam acessadas.

Além disso, Sequence também poderá ser utilizada em consultas Linq, semelhante a Sequence para consultas, onde é feita a referência para System.Linq.IQueryable utilizando expressões lambda que serão usadas e transformadas em uma expressão em árvore para serem transportadas por uma rede. A **Listagem 14** demonstra o uso de Sequence com Linq. A consulta utiliza Linq para acessar a tabela clientes e realizar um filtro através da pesquisa por nome de cliente.

Sequence vai além, possuindo suporte a paralelismo, um novo recurso disponível para utilização fazendo uso de programação paralela onde a execução ocorre em várias Threads de chamada. O código da **Listagem 15** é um exemplo de utilização de programação paralela. A definição da variável resultado utiliza funcionalidades em paralelo que estão disponíveis em System.Linq.IParallelEumerable, um tipo específico para estas representações e estilo de programação.

Parallel

O Delphi Prism apresenta uma sintaxe que possibilita utilizar paralelismo em estruturas de repetição for, condicionando elas a execução em paralelo. Dessa forma os elementos presentes na estrutura em paralelo são distribuídos

Listagem 14. O uso de sequence em consultas com Linq

```
var
  resultado: queryable sequence of Clientes :=
  from x in Tables.Clientes where x.Nome = 'Rodrigo' select x;
```

Listagem 15. Utilização de paralelismo com sequence

```
var resultado: parallel sequence of Int32 :=
   Enumerable.Range(0, 1000).AsParallel();
```

Listagem 16. Exemplo de utilização de for em paralelo

```
for parallel i: Integer := 0 to 10 do
begin
end;
```

Listagem 17. Utilizando uma estrutura for each em paralelo

```
for each parallel elemento in ColecaoObjetos do
begin
end;
```

Listagem 18. Código para demonstrar o uso de where em exceções

```
try
  // código...
except
  on E1: SocketException where E1.Errorcode = 10054 do begin
     MessageBox.Show("mensagem personalizada 1");
  end;
  on E2: SocketException where E2.Errorcode = 10055 do begin
     MessageBox.Show("Mensagem personalizada 2");
  end;
  on E3: Exception do begin
     MessageBox.Show("Mensagem genérica");
  end;
end;
```

Nota do DevMan

A classe System.IO.FileInfo fornece informações sobre um arquivo no sistema de arquivos. Além disto, a classe contém métodos de instância para criar, copiar, excluir e abrir arquivos. O construtor FileInfo(arquivo: string) inicializa uma instância da classe FileInfo como um "wrapper" para o arquivo. A propriedade Length obtém o tamanho do arquivo, em bytes, a propriedade Extension obtém a extensão do arquivo e a propriedade Name obtém o nome de arquivo do arquivo.

pelas Threads e o processo de repetição irá aguardar, sem chegar ao seu final, até que todos os elementos individuais completem suas tarefas. Caso ocorra alguma exceção, toda a estrutura de repetição é cancelada e seus elementos distribuídos pelas Threads também irão cancelar sua execução retornando o processo ao contexto original de onde foi iniciado.

Além disso, o uso de condições forçadas de saída como a utilização de break não são permitidas com parallel, pois este tipo de situação simplesmente irá terminar a execução dos elementos que estão realizando alguma tarefa, mas os demais elementos presentes poderão ainda não ter iniciado seu processamento, seguindo a sua execução.

O código da **Listagem 16** demonstra o uso de uma estrutura de repetição em paralelo. O código realiza uma estrutura de repetição simples em paralelo. Observe que a utilização de downto não é permitida e não apresenta um sentido claro, visto que a ordem de execução das tarefas não é garantida, o que poderia gerar inconsistências de programação sem resultados precisos.

Para a utilização em uma estrutura do tipo for each, também é possível executar paralelismo. Na **Listagem 17** é possível verificar a sua utilização em paralelo. Para a utilização do exemplo é necessário ter definida a coleção de objetos que será usada, neste caso, implementando uma interface genérica do tipo IEnumerable.

Where

Sua utilização possibilita realizar filtros delimitando operações e permitindo realizar tratamento diferenciado para cada filtro. Where pode ser utilizado em construções personalizadas de tratamento de exceções de código.

Na **Listagem 18** é verificado seu tratamento em um exemplo simples de exceções. O uso de where permite realizar filtros e definir padrões de mensagens mais amigáveis para quem irá utilizar o aplicativo que está sendo desenvolvido. No código da é possível observar seu comportamento definindo mensagens personalizadas para cada tipo de erro encontrado e, além disso, uma mensagem padrão de exceção quando um erro encontrado não possuir tratamento adequado.

O uso de where poderá ser acoplado a expressões de consulta com o mesmo objetivo de filtrar os dados em busca de resultados mais precisos. Expressões de consulta em Delphi Prism utilizam as definições baseadas em operações de consulta padrões introduzidas pelo .Net Framework 3.5 e seus tipos definidos no namespace System.Query. As expressões utilizam uma sintaxe de linguagem semelhante ao SQL, assim, permitindo que as consultas sejam fáceis e intuitivas para quem irá criá-las.

Por exemplo, na **Listagem 19** é obtida uma relação com todos os arquivos executáveis da pasta C:\Windows\System32 com tamanho superior a 1 MB, classificados em ordem crescente do tamanho do arquivo. Neste caso, a cláusula where permite filtrar elementos da fonte de dados baseada em uma ou mais expressões booleanas separadas pelos operadores lógicos and (e) ou or (ou).

Listagem 19. Consulta utilizando filtragem de resultado

```
from arquivo in Directory.GetFiles('c:\windows\system32')
with infoArquivo := new FileInfo(arquivo)
with tamanhoArquivoMB := infoArquivo.Length / 1024 / 1024
where (tamanhoArquivoMB > 1) and (infoArquivo.Extension.ToUpper() = '.EXE')
order by tamanhoArquivoMB
select new class (Nome:=infoArquivo.Name, TamanhoMB:=tamanhoArquivoMB);
```

Listagem 20. Definição de um tipo nested em um projeto

```
type
  Email nested in TPessoa = public class
end;
```

Nested

Um nested é um tipo definido no âmbito ou membro de outro tipo, são muito úteis para encapsular detalhes de implementação de um tipo como uma coleção, pois eles conseguem ter acesso a estados privados dos objetos. O Delphi Prism apresenta suporte ao tipo nested, geralmente utilizado em classes ou registros, são declarados acrescentando a palavra nested seguido do tipo pai e somente pode ser definido no mesmo projeto base.

A **Listagem 20** indica um exemplo básico de sua definição em um projeto. No código descrito a definição da classe TPessoa deverá ter sido realizada anteriormente para que seja possível criar um tipo nested, neste caso, para o uso de e-mail.

As regras de acessibilidade a tipos nested são aplicáveis para acesso a variáveis (campos, propriedades ou métodos) de sua classe de container, mas devem possuir referência de objeto para realizá-las. Nesse caso, a classe aninhada, só poderá ser instanciada e acessada a partir da classe que a derivou. Vale observar que deve ser evitada a declaração de tipos nested como públicos, a não ser que ocorra uma razão justificável para realizá-la. Os tipos nested são bastante utilizados em toda a estrutura do ambiente de desenvolvimento .NET e na programação orientada a objetos, apresentando uma forma de manter unidos os tipos conceituais relacionados e evitar conflitos.

Conclusão

Ao verificar as novas possibilidades e melhoramentos de linguagem presentes

no Delphi Prism é possível visualizar que elas vieram para ampliar os conhecimentos de quem as usa e melhorar o desempenho dos aplicativos que as utilizam. No decorrer deste artigo foram citadas algumas das principais melhorias, o uso de loop, case, ensure, locking, invariants, futures, for e for each acrescidos de parallel, nullable, sequence, where e nested. Como pôde ser visto, a linguagem do Delphi Prism foi melhorada, possibilitando melhor desenvolvimento de aplicações, algumas limitações foram supridas e refinamentos foram realizados com o objetivo de facilitar a vida de quem irá criar aplicativos. Diversas das ideias transmitidas no artigo poderão ser utilizadas no desenvolvimento, caso exista a possibilidade de utilização da programação paralela.

Rodrigo Araujo
rodrigo.arsi@gmail.com
Bacharel em Ciência da Computação pela UFSM – Santa Maria/RS trabalha com Análise e Desenvolvimento de Software há mais de 10 anos. Desde 2003, utiliza tecnologias .NET.

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Cursos Online

Assinatura .net magazine PLUS
Mais conteúdo .NET por muito menos!

A Revista **.net Magazine** oferece para seus assinantes uma série de Cursos Online de alto padrão de qualidade .

Conheça abaixo os cursos já disponíveis:
www.devmedia.com.br/curso/netmagazine

- **Crie uma loja Virtual completa**
- **Construindo relatórios com Crystal Reports e Visual Studio 2005**
- **Criando uma aplicação Web Completa**
- **Criando uma aplicação client/server no Visual Studio 2005**
- **Aprenda a criar um blog com ASP.NET**
- **Curso de C#**

A sua melhor opção de aprendizagem!
Assine a **.net Magazine** e Comece já seu treinamento!
www.devmedia.com.br/assine

DevMedia Mais Informações: www.devmedia.com.br/central - Tel.: 21 3382-5038

RENOVE JÁ!

Sua assinatura pode estar acabando

Renovando a assinatura de sua revista favorita você ganha brindes e descontos exclusivos.

Faça a renovação de sua assinatura agora mesmo.

www.devmedia.com.br/renovacao ou (21)3382-5038

DevMedia

MadShi Collection
Tratamento de exceções e hooking

Existem certas bibliotecas para o Delphi tão úteis que deveriam vir com o produto. A política da Borland/Codegear/Embarcadero sempre foi colocar no mercado a ferramenta com os recursos essenciais e permitir que desenvolvedores de componentes e experts desenvolvam extensões para o IDE. Isso é bom para as empresas que desenvolvem componentes, mas não tão bom, talvez, para as empresas que desenvolvem software customizado.

A MadShi Collection é uma coleção de bibliotecas que vem com várias utilidades, dentre elas a MadShiExcept e a MadShiCodeHook. No pacote todo, algumas das suas funcionalidades são gratuitas e com fonte, algumas são gratuitas para uso não comercial e sem fontes, e algumas são apenas de uso por tempo limitado (trial) sem fontes. Mesmo assim a biblioteca inteira pode ser adquirida com fontes e licença para desenvolvedores ilimitados.

É necessário avaliar o custo-benefício. Se uma empresa tem mais de 4 programadores e um número razoável de clientes é possível comprar a MadShiExcept por 449 Euros, para um número ilimitado de programadores e com código fonte.

A biblioteca MadShiCodeHook é um pouco mais cara, podendo chegar a 1999 Euros dependendo da licença. Mas ela permite criar softwares bastante fora do comum, por exemplo interceptando e alterando os parâmetros de chamadas a funções de outros softwares ou da API do Windows.

A grande vantagem de se trabalhar com a MadiShiExcept é a clareza e transparência que ela produz ao exibir erros para o usuário. O usuário sabe que pode enviar para seu fornecedor um relatório de erros para que seja corrigido, e sabe que o relatório desse erro ficou gravado em algum lugar. Nada de tirar print screen da tela, soletrar palavras em inglês técnico, que muitas vezes são incompreensíveis até para quem tem inglês fluente e sem ter que lembrar o que foi feito de errado para o erro ocorrer. Simplesmente é retirado das costas do usuário esse "sentimento de culpa" que pode colocá-lo na defensiva (ou ofensiva)

Resumo DevMan

De que trata o artigo:

Este artigo será um estudo dirigido da biblioteca MadShi. Serão mostrados vários aspectos da biblioteca, como as ferramentas de gerenciamento de exceções e API hooking. Os exemplos práticos envolverão a apresentação de um relatório avançado de erros com snapshot automático e informações de debug. O exemplo prático de API hooking mostrará um monitor de arquivos.

Para que serve:

A biblioteca MadShi tem vários objetivos, dentre eles Exception Handling e API Hooking. Exception Handling serve para obter informações úteis e concisas sobre erros não tratados. Isso pode ajudar muito a agilizar a correção de bugs e o bug tracking. Além disso, o usuário se sentirá mais seguro com uma mensagem de erro amigável e a possibilidade de enviar um relatório para seu fornecedor. Já API Hooking é essencial para interceptar e modificar funções de DLLs de outros programas ou até do próprio Windows.

Em que situação o tema é útil:

Em casos de programas com muitas exceptions não tratadas e muitas mensagens de "Invalid Pointer Operation" e "Access Violation" é importante saber exatamente em que linha e unit o problema se encontra, sem precisar debugar o programa ou levar o Delphi para o cliente. API hooking é útil para integração com outros softwares ou para segurança. É possível impedir o desligamento do servidor, impedir a exclusão ou abertura de arquivos essenciais ao sistema, monitorar tudo o que é digitado (Keylogger), proibir a abertura de certos programas dentre muitas outras utilidades.

Exceções e API Hooking:

Com certeza você já se deparou com um erro de Access Violation ou Invalid Pointer Operation durante a demonstração de um sistema feito em Delphi. Essas mensagens, além de feias, são alarmantes para o cliente, e constrangedoras para o desenvolvedor. E por mais que se tente usar logs e tratamentos de exceções mostrando qual a janela aberta e como a exceção foi disparada, no máximo terá o controle que disparou o erro, caso ele tenha sido disparado por um controle. O programador muitas vezes tem que adivinhar em que linha que ocorre o erro ou debugar o programa com uma base de dados real para poder encontrar e corrigir esse bug. Com relação à segurança, utilizando API hooking é possível implementar funções básicas de um sistema de Cyber Café / Lan House sem a necessidade de um software muito caro para isso. Como este artigo é um estudo dirigido será mostrada uma quantidade maior de exemplos teóricos do que sistemas propriamente ditos.

de API e código

VITOR LUIZ RUBIO

ao se tentar questionar como o erro ocorreu. Ocorreu um erro? A responsabilidade de saber como foi que isso aconteceu é da equipe de desenvolvimento.

Por outro lado a biblioteca MadShiExcept oferece transparência também do lado do desenvolvedor. É possível obter informações completas sobre como o erro foi gerado, inclusive nome da unit e linha onde ocorreu o erro. E ainda é possível mesclar essas informações com outras sobre o ambiente em que o programa estava sendo executado. Armazenando-se essas informações em um banco de dados pode-se ter um sistema de bug tracking unificado para todos os sistemas e clientes.

O primeiro passo é baixar e instalar a biblioteca MadShi. O endereço para download encontra-se na seção de links desse artigo. A **Figura 1** mostra as opções que foram instaladas para este artigo.

Exceptions

Pode-se dividir o conjunto de exceções de um sistema em dois subconjuntos: exceções tratadas, que são aquelas que são esperadas e até desejadas e que por serem conhecidas ocorrem dentro de um try .. except para prover uma mensagem amigável ao usuário e talvez até uma saída alternativa. É comum este try..except estar dentro de um try.. finally maior, para liberar recursos que de outra forma não poderiam ser liberados dentro de um except. A instrução finally sempre é executada, mesmo fechando-se o programa, caso o fluxo esteja dentro de um try, o finally será executado.

O outro grupo de exceções é o grupo das exceções não tratadas, onde a princípio não se sabe como e porque elas ocorrem, e talvez apenas o usuário saiba fazer a "mágica" acontecer, mas não saiba explicar como, e nem deveria, pois na concepção dele está fazendo tudo corretamente.

A **Listagem 1** exibe primeiramente uma exceção não tratada e na sequência uma tratada. Na exceção tratada, após o except, o fluxo continua normalmente. Na exceção não tratada o fluxo é interrompido depois do disparo da exceção, e a linha logo abaixo não é executada.

Geralmente aqui pode ocorrer um Access Violation com endereço 0x000... se o problema foi causado por acesso a um objeto não criado (nil) ou algo diferente de zero para acesso a um objeto já destruído (diferente de nil mas inválido). Podem também ocorrer erros de IO, acesso a banco, conexão com a internet, transmissão, estouro de capacidade de pilhas, variáveis (integer overflow), algum índice out of bounds, stack overflow e até uma Falha Catastrófica. Tudo isso confunde muito o usuário, assustando-o e não diz nada ao programador. Não diz onde ele errou e onde tem que consertar. A não ser que ele seja muito bom de debug, e usando um debugger

Listagem 1. Exemplo de exceptions tratadas e não tratadas

```
procedure TfrmDemo.Button1Click(Sender: TObject);
begin
  raise Exception.Create('MissUnderstoodError at adress
  0x00000000 :$');
  ShowMessage('Linha logo depois de ter ocorrido a exception.');
end;

procedure TfrmDemo.Button2Click(Sender: TObject);
begin
  try
    raise Exception.Create('MissUnderstoodError at adress
    0x00000000 :$');
  except
    on e: Exception do
    begin
      ShowMessage('Ocorreu o seguinte erro: ' + e.Message);
    end;
  end;
  ShowMessage('Linha logo depois de ter ocorrido a exception.');
end;
```

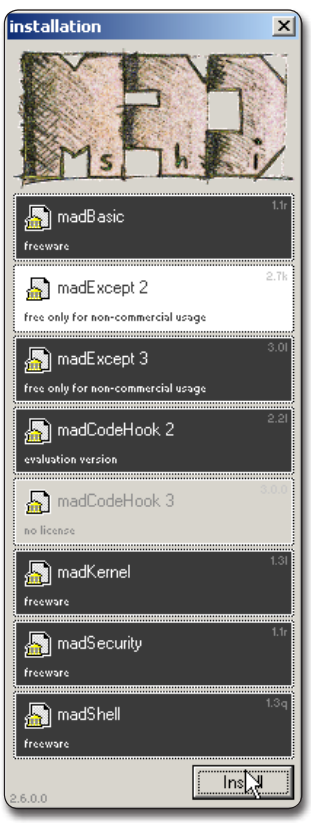


Figura 1. Instalador da coleção MadShi

Nota

Para facilitar o estudo e prática desses exemplos opção Notify on Language Exceptions foi desmarcada, (Menu Tools>Options>Embarcadero Debuggers>Language Exceptions). Executar o projeto através da opção Run Without Debugging também gerará o mesmo efeito. Isso fará com que não sejam vistas as mensagens de exceptions do debugger, mas ainda é possível ver as mensagens de exception do próprio programa. Isso ajuda a não parar o debugger com um break a cada exception, mas ainda assim permite que o código seja executado passo a passo.

Nota

Antes de aprender a usar a MadExcept é bom conhecer as novas propriedades das Exceptions do Delphi XE. Algumas propriedades como StackTrace, InnerException entre outras não estavam presentes no Delphi 7, por exemplo.

externo ao Delphi ele observe o código ASM e o endereço do erro e perceba o que está dando errado.

Existem muitas pessoas que encontram bugs assim, e são admiráveis. Para o restante é necessário criar um gerenciador de exceções genérico que possa obter o máximo possível de informação útil, e talvez um log. Nesses casos o que geralmente se faz é colocar um TApplicationEvents no formulário principal ou no datamodule, a primeira coisa que for criada, e fazer o tratamento genérico ali no evento OnException.

Listagem 2. Exemplo para tratar exceções

```
class procedure TfrmExceptions.ShowError(sender: TObject; e:
Exception);
var
 quemGerou: string;
begin
 with TfrmExceptions.Create(nil) do
 try
  try
   mErro.Clear;
   mErro.Lines.Add('Erro: ' + e.Message);
   mErro.Lines.Add('Tipo de Erro: ' + e.ClassName);
   mErro.Lines.Add('-------------------------------');
   if sender is TComponent then
    quemGerou := (sender as TComponent).Name
   else
    quemGerou := sender.ClassName;

   mErro.Lines.Add('Quem gerou: ' + quemGerou);
   mErro.Lines.Add('Classe que gerou: ' + sender.ClassName);
   mErro.Lines.Add('Unit que gerou: ' + sender.UnitName);
   ShowModal;
  except
  end;
 finally
  Release;
 end;
end;
```

Listagem 3. Exceptions encadeadas depois

```
procedure TfrmAntesDepois.btDepoisClick(Sender: TObject);
begin
 try
  try
   raise Exception.Create('Agora duas exceptions aninhadas
   '+ #13#10);
  except
   on e: exception do
   begin

     e.ThrowOuterException(Exception.Create(
      'Podem ter suas mensagens recuperadas separadamente
       graças ' + #13#10+
      'ao mecanismo de InnerException. E não apenas a
       mensagem, mas '+ #13#10 +
      'todo o objeto exception com todas as suas informações.'
     ));

   end;
  end;
 except
  on e: Exception do
  begin
   MessageBox(Handle, pchar(e.InnerException.Message), 'Erro',
   MB_ICONWARNING);
   MessageBox(Handle, pchar(e.Message), 'Erro', MB_ICONWARNING);
  end;
 end;
end;
```

A **Listagem 2** exibe um trecho de código que exemplifica o tratamento genérico de exceções. O código inicia instanciando um formulário que contém alguns controles que exibem o erro detectado. O texto do erro é formatado para uma forma mais didática. Inclusive essa formatação também é feita dentro de um bloco try..except porque caso algo dê errado ocorrerá um erro stack overflow, porque a rotina para captura de erros chamará ela mesma novamente, entrando em um loop infinito. Contudo a abordagem é frágil porque só mostraria a unit e o nome do componente correto se o componente estiver no formulário, fora de um GroupBox, Panel ou qualquer outro container. Caso esteja dentro de um container, a unit, nome e nome de classe que vão aparecer são os do container e não da form.

Pode-se melhorar isso utilizando a API do Windows. Obtendo a janela em foco e a posição X e Y do mouse e assim saber onde o usuário clicou, se é que a exceção foi disparada por um clique em um controle. E se ela for gerada por um componente de acesso a dados?

Esse código é todo baseado em uma class procedure. Essa procedure tem a responsabilidade de criar uma instância do formulário de tratamento de exceções, exibi-lo de maneira modal e destruí-lo. A assinatura dessa procedure é a mesma e o evento OnException do objeto Application. Sendo assim é possível usar um evento OnException de um TApplication (contanto que ele esteja na primeira unit ou datamodule) ou simplesmente no arquivo DPR fazer Application.OnException := TfrmExceptions.ShowError;

De qualquer forma não é possível ainda obter a unit real e o número da linha onde ocorreu a exceção. Uma outra melhoria natural seria um loop para armazenar informações das exceptions anteriores (inner exceptions).

Exception Chains

Uma nova possibilidade nas versões novas do Delphi é o uso de exception chains. Caso se esteja programando em uma camada intermediária, e ocorrer uma exceção em uma camada mais baixa, é possível disparar uma outra exceção na camada intermediária contendo informações adicionais e contendo como InnerException (Exceção interna) a exceção que já ocorrera na camada mais baixa. Com isso é possível repassar uma exceção para as camadas superiores sem necessariamente ter que criar uma nova do zero ou capturar a anterior apenas para concatenar mais informações na mensagem. Além de facilitar na adição de informações a uma exceção não é necessário artifícios como ignorá-la ou dispará-la novamente. E é possível saber exatamente em que camada a exceção ocorreu e porque a próxima camada não pode lidar com isso. Para isso usa-se o método ThrowOuterException, mostrado na **Listagem 3**.

Antigamente, em uma sequência de exceptions era necessário coletar as mensagens para apresentar ao usuário todas juntas, ou registrar um log de todas juntas. Mas as outras informações da exception eram perdidas. Hoje várias exceptions podem ser encadeadas sendo disparadas

umas dentro das outras. No exemplo visto primeiramente foi tratada a exceção interna, exibindo sua mensagem e informações, depois a exception externa. Podem existir várias InnerExceptions ligadas em cadeia, onde cada exception possui uma InnerException. A ordem de tratamento, registro de log ou exibição pode ser diferente de acordo com cada caso.

MadExcept

Muitos softwares feitos em Delphi ou C++ Builder utilizam a biblioteca MadShi. Uma das grandes vantagens da MadExcept é o fato de que ela é thread-safe. Então caso seja utilizado em um sistema multithread e ocorra uma exceção, é possível isolá-la das outras threads e saber exatamente em que thread, unit e linha a exceção ocorreu. Além disso, todas as caixas de diálogo do MadExcept são feitas utilizando-se Windows API pura, o que garante, além da segurança inter-threads, que tudo funcione corretamente inclusive em programas não VCL. Se houver algum bug na VCL que disparasse uma exception, o fato da MadExcept não usar a VCL garante que as exceções não sejam disparadas em loop infinito pelo seu próprio gerenciador.

Dizer simplesmente que a biblioteca madExcept mostra o número da linha e a unit onde ocorreu a exceção é subestimar o pacote. Na verdade essa biblioteca é poderosa o suficiente para auxiliar na busca por exceptions geradas na seção de inicialização das units, geradas dentro de dlls ou geradas em qualquer ponto ou programa em que não se tenha a VCL. Em um relatório de exceções é possível colocar um dump da pilha do sistema, um snapshot da tela onde o erro foi gerado, a lista de processos rodando, a lista de módulos do sistema carregados, o nome do computador, o nome do usuário, toda call stack, memória livre e alocada, espaço livre em disco, valores dos registradores da CPU, detalhes do hardware e até o trecho que gerou a exceção em Assembly.

Utilizar o madExcept é muito simples e não requer nenhuma alteração nos programas, inclusive nos legados. Basta apenas adicionar certas units no início de todo o projeto, logo no uses do arquivo .dpr ou utilizar um assistente localizado em Project>madExcept settings. Ao abrir o assistente uma série de configurações serão apresentadas, estão categorizadas em abas (guias) e para ativar o uso do madExcept deve-se habilitar o checkbox enable madExcept. Na primeira guia, "basic settings", a opção "check for frozen main thread" verifica se o programa está recebendo mensagens do Windows. Caso não esteja então o programa está congelado, pode ser um loop infinito ou um deadlock, ele disparará uma mensagem

Nota

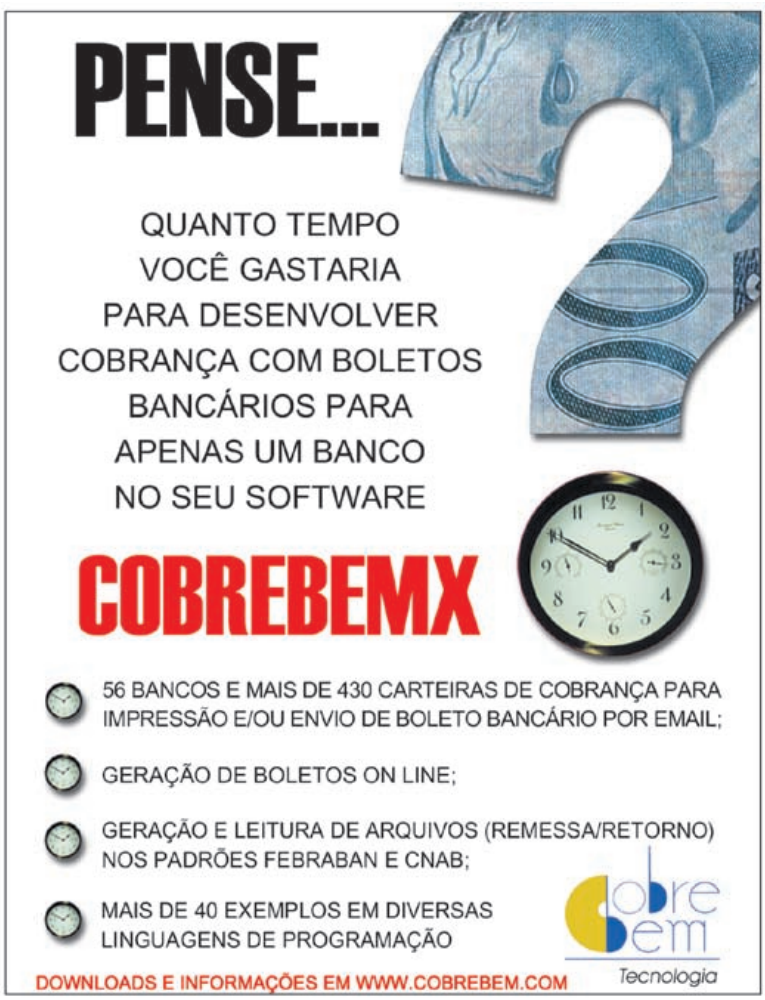
Está disponibilizado para download um programa completo para testar um gerenciador de exceções e o gerenciador propriamente dito. É o mais longe que se pode chegar sem usar bibliotecas de terceiros.

Nota do DevMan

Embora pouco documentadas as funções ThrowOuterException e RaiseOuterException encontradas na unit SysUtils são idênticas tanto na assinatura quanto na implementação, e têm o mesmo propósito: disparar uma nova exception contendo a exception anteriormente disparada na camada mais baixa como sua InnerException. Tanto ThrowOuterException como RaiseOuterException são class procedures e são implementadas com o código:

```
class procedure Exception.RaiseOuterException(E: Exception);
asm
   TEST  EAX,EAX
   JZ    System.@RaiseExcept
   MOV   [EAX].Exception.FAcquireInnerException,1
   JMP   System.@RaiseExcept
end;

class procedure Exception.ThrowOuterException(E: Exception);
asm
   TEST  EAX,EAX
   JZ    System.@RaiseExcept
   MOV   [EAX].Exception.FAcquireInnerException,1
   JMP   System.@RaiseExcept
end;
```

e será possível ver o local do loop. Na guia "on exception auto actions", a opção "copy bug report to clipboard" permanece habilitada, assim o erro vai para a área de transferência permitindo ao usuário que cole o erro no corpo de um e-mail, por exemplo. Já a opção show "please wait" é desmarcada, ela funciona como um temporizador. Já na guia "exception box settings" as opções "save bug report" e "print bug report" são marcadas para que haja a opção de salvar e ou imprimir o erro. Em "email & upload settings" pode-se configurar uma conta de e-mail que será utilizada para enviar os relatórios e por fim na guia "custom settings" pode-se traduzir as mensagens que serão exibidas pela madExcept, inclusive é possível criar um texto padrão para as exceptions. A configuração aqui sugerida foi otimizada para o caso de acontecer uma exceção numa hora que o usuário não estiver olhando para a tela (por isso foi tirado o temporizador) e para o caso de o usuário não estar com o cliente de e-mail configurado. Caso não seja possível enviar o e-mail é possível imprimir o relatório de erros.

Talvez o diálogo mais importante seja o último, onde é possível personalizar todas as mensagens e rótulos mostrados ao usuário, para que não fique em inglês. Também é possível marcar este formato como padrão (default) para que fique igual em todos os projetos da empresa.

Depois de configurado esse wizard, na prática, o que muda é o arquivo .dpr, que ganha referências a mais algumas units (madExcept, madLinkDisAsm, madListHardware, madListProcesses, madListModules).

Ao executar um programa que utiliza a madExcept é possível ver que as exceptions tratadas continuam como sempre foram, mas a exceção não tratada abre a caixa de diálogo do madExcept, que permite visualizar a mensagem de erro, enviar o relatório de erros por e-mail, imprimir esse relatório e visualizá-lo. Nessa visualização as abas General e Call Stack contêm informações importantes.

A biblioteca madExcept utiliza técnicas de hooking para interceptar certas funções e gerenciadores de erros do Delphi trocando-as por funções próprias. Por isso algumas units são adicionadas no projeto antes de todas as outras. Elas devem ser carregadas antes de qualquer outro módulo do sistema para que não ocorra nenhuma exceção antes de seu carregamento. Caso contrário, por exemplo, se a primeira unit de uma aplicação é um datamodule e as units do madExcept fossem carregadas apenas depois desse datamodule corre-se o risco de, acontecendo uma exceção relacionada a conexão com o banco de dados (caminho errado, por exemplo) esta ser tratada pelo mecanismo tradicional e não pelo madExcept que ainda não foi carregado. Antes da versão 2006 do Delphi não vinha de fábrica a biblioteca FastMM para gerenciamento rápido de memória e detecção de vazamentos, sendo assim, quando um programador usava em um projeto feito em Delphi 7 a biblioteca FastMM, a unit FastMM.pas fazia hooks de baixo nível, em assembly, para interceptar o gerenciador de memória do próprio Delphi e trocá-lo por um melhor e mais rápido. Uma técnica de fato avançadíssima.

Uma pequena desvantagem da madExcept é que ela ainda não faz uso das inner exceptions para apresentar qual exceção interna que causou a externa. Pode ser visto na **Figura 2** que apenas a exceção da procedure RethrowEmUmaExcecao é mostrada, e que ela foi chamada pelo Button1Click. De fato uma exceção "redisparada" é considerada como tratada sendo a exceção externa considerada inesperada.

Neste exemplo é mostrado pela biblioteca que uma exceção ocorreu na linha 76 (**Figura 3**), mas não mostra que esta foi disparada unicamente para alertar que outra mais interna ocorrera na linha 54. Isso poderia levar um desenvolvedor a tratar a exceção externa ou retirar o ThrowOuterException para procurar pela exceção interna, visualizando assim seu stack. É uma possibilidade. Uma opção seria não trabalhar com ThrowOuterException por enquanto.

Outra desvantagem é o fato de que dependendo de como estiver configurado o assistente do madExcept (opção para congelar ou não o programa durante a apresentação de uma exceção) o programa pode ser fechado com a janela de exceção aberta, o que causará um leak (um objeto não destruído na memória). É possível, se necessário, escrever seus próprios gerenciadores de exceções, derivados do madExcept para registro dos erros em banco de dados ou help desk e para tratamento avançado das inner exceptions.

Nota

Para enviar e-mail sem ser pelo Outlook é necessário que o firewall permita o envio de e-mail pela porta 25 para se atuar como SMPT Server e que este IP esteja liberado para relay. Para atuar como SMTP client o desenvolvedor deve se certificar de possuir um servidor de e-mail que atue na porta 25, sem SSL, pois o madExcept não dá suporte a SSL (como o SMTP do Google).

Nota

Durante a elaboração deste artigo a biblioteca madShi não suportava ainda o Delphi XE. A última versão suportada é a 2010, na qual foi elaborada este artigo.

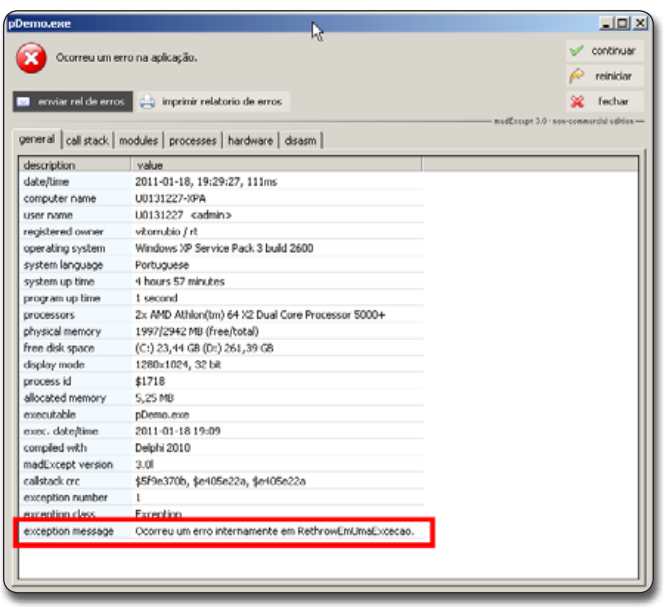


Figura 2. Mensagem da exception

API Hooking

API Hooking é a técnica de se interceptar certas funções de certos programas ou dlls. Interceptando essas funções é possível se antecipar da sua execução e rodar uma outra função substituta no lugar. Ainda é possível optar por executar a função original ou não.

Se uma função, por exemplo, MessageBoxW for interceptada é criado na pilha (stack) um encadeamento, onde o endereço de memória da função MessageBoxW pode levar a uma função MessageBoxW_Alterada1, que somente depois pode levar novamente ao endereço original, ou não. Se essa função for interceptada novamente por outra MessageBoxW_Alterada2 esta interceptará na verdade MessageBoxW_Alterada1 e a cadeia de chamadas será MessageBoxW_Alterada2 > MessageBoxW_Alterada1 > MessageBoxW(original).

Em um antivírus ou um programa de monitoramento/segurança é bem provável que se utilize algumas funções da API do Windows, como a SHFileOperation, que é usada sempre que se copia, recorta ou cola um arquivo. Se o computador onde esse antivírus é executado contiver ainda um programa de segurança (como esses usados em laboratórios de faculdades e lan houses) é possível ainda que a SHFileOperation esteja interceptada duas vezes, sendo que uma chamada a essa API passa primeiro pelo antivírus e depois pelo programa de segurança.

Caso um computador tenha algum jogo online instalado é comum que tenha também algum programa de segurança para detectar KeyLoggers, debuggers, sniffers e outros programas utilizados para se trapacear no jogo.

Outras aplicações para API Hooking, além de programas de segurança, log e monitoria, podem incluir substituir todas as chamadas a uma determinada função pela chamada a uma função aprimorada.

MadshiCodeHook

A biblioteca é bastante versátil, alguns vírus podem ser escritos com o auxílio desta biblioteca, e antivírus também. É importante deixar claro que antivírus realmente eficientes são escritos na forma de drivers que executam em modo kernel. Ou seja, são hooks de operações mais internas do sistema operacional, bem longe da camada de usuário onde a maioria dos programadores está habituada a trabalhar.

O primeiro exemplo a ser mostrado é a substituição, em um processo, de todas as chamadas a uma função por outra. Esta não é a melhor maneira de se fazer isso se for considerada a hipótese de substituir todas as linhas de código com essa chamada, usando Search e Replace. Pode ser a única alternativa caso seja um sistema legado com muitas linhas de código. Esta técnica poderia ser usada para impedir que programadores utilizem certas funções, mas também existem auditores de código para esse fim.

Uma situação interessante seria a substituição de uma função por outra mais atual, de mesmo nome ou nome diferente, quando um mesmo sistema utiliza versões diferentes da mesma DLL ou BPL. De qualquer forma é bom ter em mente que um Hooking é um hack. É um desvio de código onde são inseridas instruções em uma rotina que não foram intencionadas pelo programador original.

Hooks mal feitos podem fazer o Windows travar, reiniciar ou coisa pior. Por isso certas definições devem ser explicadas e certas regras devem ser obedecidas:

• Alvo (ou target) é a função que será "grampeada" ou interceptada;
• Callback é a função desenvolvida para substituir a função alvo;
• Next é um ponteiro para função com a mesma assinatura que a função original que receberá seu endereço de memória. Dentro de um callback não pode-se invocar a função original, pois essa chamada se tornará um loop recursivo infinito e acontecerá um stack overflow. Para executar a função original chame pela Next;

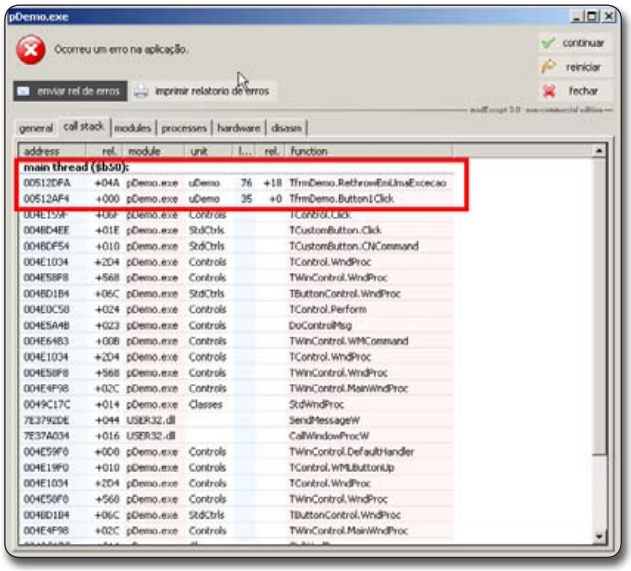


Figura 3. Call Stack

Nota

Para minimizar a possibilidade de se criar vírus com a madCodeHook a versão de demonstração, além de precisar de uma DLL e um executável para funcionar, não pode interceptar algumas funções predefinidas, como as que listam os processos no gerenciador de tarefas. Se estiver usando a versão trial do madCodeHook deve-se colocar a DLL madCHook.dll na pasta System32 do Windows e executar o programa mchEvaluation.exe com privilégios de administrador. Ambos encontram-se no diretório C:\Arquivos de programas\madCollection\madCodeHook\Dll. O programa mchEvaluation.exe da última versão estável pode não funcionar no Windows XP Service Pack 3, mas a versão BETA nova funciona. Há um link na seção de links para download da nova versão BETA que não aparece no site, mas foi publicado no fórum do madCollection.

• Grampear(hooking) será o termo utilizado para evitar um anglicismo incorreto como "hookar";
• Usar um padrão de nomenclatura: Se a função a ser grampeada se chama "batatinha", então a callback se chamará "batatinhaCallback" e o ponteiro para a "batatinha" original se chamará "batatinhaNext";
• Declarar stdcall no final de callbacks e nexts se estiver grampeando funções da API do Windows;
• Sempre usar o mesmo número e os mesmos tipos de parâmetros da função alvo, bem como seu retorno, tanto no callback como no next;
• A função de callback deve ser o mais minimalista possível;

• Evitar usar classes, funções e procedures da VCL dentro de callbacks destinados a grampear funções da API do Windows porque a maior parte da VCL não é thread-safe;

Figura 4. Interceptando MessageBoxW

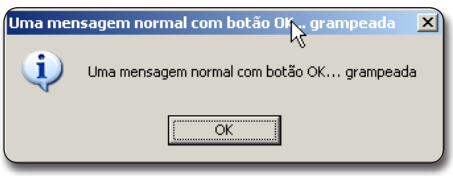


Figura 5. MessageBox grampeada

Listagem 4. Grampeando a API MessageBoxW

```
var MessageBoxWNext : function (
  handle: HWND;
  mensagem: pchar;
  caption: pchar;
  uType: cardinal) : integer; stdcall;

function MessageBoxWCallback(
  handle: HWND;
  mensagem: pchar;
  caption: pchar;
  uType: cardinal) : integer; stdcall;

implementation

function MessageBoxWCallback(
  handle: HWND;
  mensagem: pchar;
  caption: pchar;
  uType: cardinal) : integer; stdcall;
begin
  result := MessageBoxWNext(
    handle, //manter o mesmo
    pchar(mensagem + '... grampeada'),
    pchar(caption + '... grampeada'),
    MB_ICONINFORMATION
    );
end;

{$R *.dfm}

procedure TfrmExemplo.Button1Click(Sender: TObject);
begin
  MessageBoxW(Handle, 'Uma mensagem normal com botão OK',
  'mensagem', MB_OK);
  MessageBoxW(Handle, 'Uma mensagem de erro com botão
  abort e retry', 'mensagem', MB_ABORTRETRYIGNORE or
  MB_ICONERROR);
end;

procedure TfrmExemplo.Button2Click(Sender: TObject);
begin
  HookAPI('user32.dll', 'MessageBoxW', @MessageBoxWCallback,
  @MessageBoxWNext);
end;

procedure TfrmExemplo.Button3Click(Sender: TObject);
begin
  UnhookAPI(@MessageBoxWNext);
end;
```

• As funções terminadas em W da API do Windows foram feitas para lidar com Widechar0s, enquanto que as terminadas com A lidam com AnsiChars. O Char padrão do Delphi a partir da versão 2009 é WideChar, isso merece atenção;
• Evitar bloquear a função original a ser chamada, deixar que o código original execute, se possível;
• Grampos para processos não podem trabalhar com mensagens ou janelas, ShowMessage, MessageBox, PostMessage e FindWindow, tudo o que trabalha com janelas, controles e handles é proibido;
• Evitar criar threads e janelas em uma HookDll a menos que seja estritamente necessário;
• Evitar chamar VirtualAlloc pois isso pode fazer com que processos de 16 bits travem;
• Evitar mudar o valor do LastError com as APIs GetLastError ou SetLastError.

A **Listagem 4** mostra uma interceptação da API MessageBoxW (a MessageBox padrão do Delphi 2010). O parâmetro handle é mantido original, o caption e o text são concatenados com a palavra "...grampeado..." e o último parâmetro, que indica quais são os ícones e botões (uType) é totalmente ignorado e substituído por MB_ICONINFORMATION onde só aparecerá o ícone de informação (branco e azul com a letra "i") e o botão OK. São usadas duas mensagens de ícones e botões diferentes no teste, mas todas ficam iguais depois de interceptadas. O formulário de teste possui três botões: um para mostrar as mensagens, um para ligar o "grampo" e outro para desligá-lo conforme é mostrado pela **Figura 4**. O resultado é visto na **Figura 5**.

O procedimento para interceptar uma API é sempre o mesmo: criar uma variável do tipo procedure (Next) com a assinatura do método a ser interceptado, que receberá um ponteiro para a função original. Depois criar a sua função personalizada (callback) que pode ou não executar a next internamente, e por último deve-se executar a função HookAPI com os parâmetros: nome da dll, nome da função a ser grampeada, endereço do seu callback (por isso o uso de @) e endereço do método next.

Esse teste funciona apenas para esta aplicação, para funcionar para todos os processos do Windows o código deve ser feito em uma DLL e um outro programa deve "injetar" essa DLL nos processos do Windows. O método para essa injeção é o InjectLibrary. É preciso ter cuidado e se possível realizar os testes em uma máquina virtual, pois se erros acontecerem o sistema irá reiniciar. Alguns antivírus podem atrapalhar o processo de criação e depuração de programas que fazem API Hooking, por isso eles podem ser desabilitados temporariamente.

O próprio processo de debug de qualquer IDE como o Delphi já é por si só um processo invasivo, pois age em uma camada mais baixa do sistema para interceptar um programa inteiro e fazê-lo executar uma instrução por vez e monitorando todos os valores internos. Por isso algumas aplicações que se utilizam de API Hooking podem ocasionar um crash caso rodem a partir do debug do Delphi, mas podem não ter nenhum efeito colateral se compiladas

Nota

Algumas dessas regras foram adaptadas livremente do artigo Hooking Rules do help do madCodeHook, outras foram simplesmente acrescentadas neste artigo. Sinta-se livre para desobedecer todas essas regras.

Nota

Para usar e interceptar APIs do Windows 64 bits é necessária a versão paga da biblioteca, mais o carregamento de um driver (requer assinatura/certificado digital de código) e umas duas linhas de código a mais.

Nota

Junto com o download do artigo está disponibilizado o código fonte mostrando como criar a DLL ExemploDLL.dll.

e executadas por fora do debug (ou com shift + ctrl + F9).

Para criar um grampo para a API MessageBoxW que funcione em todas as aplicações é necessário criar uma DLL, recortar o código do callback e do next mostrados anteriormente e introduzir dentro dessa DLL.Então é necessário criar um programa que poderá incluir esse grampo ou removê-lo do sistema operacional. Os códigos que incluem e retiram são vistos a seguir:

```
InjectLibrary  (ALL_SESSIONS or SYSTEM_PROCESSES,
'ExemploDLL.dll');
UninjectLibrary  (ALL_SESSIONS or SYSTEM_
PROCESSES, 'ExemploDLL.dll');
```

Para testar pode ser usado o bloco de notas. Ao abri-lo, digita-se qualquer coisa para então fechá-lo. Nesse momento ele usará a API MessageBoxW para perguntar se é necessário salvar o arquivo ou não. Após executar a injeção da biblioteca a mensagem para salvar o arquivo ficará conforme a **Figura 6**, mostrando que agora o grampo funcionará para todo e qualquer processo que use a API MessageBoxW.

O método UninjectLibrary faz exatamente o contrário do InjectLibrary: descarrega a DLL-grampo. Os parâmetros ALL_SESSIONS or SYSTEM_PROCESSES significam que todos os processos serão grampeados, inclusive os de sistema. São flags que podem ser ativadas ou desativadas mediante o operador OR. A lista das flags e seus efeitos são:

- ALL_SESSIONS - Aplica-se a todas as sessões;
- CURRENT_SESSION- Aplica-se a sessão corrente (não sobrevive a um logout e login);
- SYSTEM_PROCESSES - aplica-se a processos de sistema e serviços;
- CURRENT_PROCESS - retira o grampo do próprio programa que injetou a DLL. Serve para apenas o programa que injetou a DLL poder chamar as APIs "puras".

Para fazer com que funcione o grampo para todas as aplicações exceto para seu programa deve-se injetar para tudo e ejetar apenas de seu processo, como mostrado na seguinte instrução:

```
InjectLibrary  (ALL_SESSIONS or SYSTEM_
PROCESSES, 'ExemploDLL.dll');
UninjectLibrary  ((ALL_SESSIONS or SYSTEM_
PROCESSES) or (not CURRENT_PROCESS),
'ExemploDLL.dll');
```

Interceptando código

Além de interceptar chamadas à DLLs também é possível interceptar código do próprio programa e modificar o destino de chamadas a métodos estáticos globais. Por exemplo, é possível fazer com que toda chamada à ShowMessage se transforme em uma chamada a um ShowMessageEspecial que pode ser desde um formulário com uma mensagem temporizada até um MessageBox.

Para isso usa-se a função HookCode em vez de HookApi. Além disso, essa opção funciona apenas no programa/processo atual, de onde foi chamada. O código da **Listagem 5** mostra como utilizar a função Hookcode.

Monitorando operações com arquivos

Até agora os exemplos foram de interceptação de funções que disparam mensagens. Não foi dada muita atenção às funções terminadas em "A", de AnsiString/AnsiChar, mas dessa maneira qualquer programa que foi compilado utilizando as bibliotecas "A", como os compilados no Delphi 7, não terão suas mensagens interceptadas.

Criar grampos de API sempre envolverá repetição de código para se criar as funções de callback correspondentes para o grupo "A" e para o grupo "W". Uma regra prática é pesquisar pela API que será grampeada no site MSDN. Lá é possível verificar as entradas e saídas das funções.

Além de verificar a função no MSDN é interessante também verificar no Delphi, na unit onde a função está declarada (Windows.pas, ShellApi.pas ...), é bom para saber como ela está declarada. Se o parâmetro for const, a função de Callback deve ter também um parâmetro const.

Uma regra básica é não esquecer da diretiva stdcall no final da declaração das funções de Callback e Next. Toda a API do Windows usa a convenção de chamada stdcall. Seu não uso faz com que Windows faça um reboot logo após o hook.

Uma das funções a serem grampeadas é a SHFileOperation, nas versões W e A. Essa é a função da DLL Shell32.dll que é usada quando se trabalha com os arquivos utilizando o Windows Explorer. Existem outras funções na kernel32.dll que precisam ser vistas, dentre elas:

- CopyFile;
- CopyFileEx;
- CreateFile;

Listagem 5. Grampeando código de procedures locais

```
var
  ShowMessageNext: procedure
  (const msg: string);

procedure ShowMessageCallback
  (const msg: string);

implementation

procedure ShowMessageCallback
(const msg: string);
begin
  MessageBox(0, pchar(msg), 'Mensagem do
  sistema:', MB_ICONINFORMATION);
end;

{$R *.dfm}

procedure TfrmExemplo.Button1Click
(Sender: TObject);
begin
  ShowMessage('Mensagem de Teste');
end;

procedure TfrmExemplo.Button2Click
(Sender: TObject);
begin
  HookCode(@ShowMessage,
  @ShowMessageCallback, @ShowMessageNext);
end;

procedure TfrmExemplo.Button3Click
(Sender: TObject);
begin
  UnhookCode(@ShowMessageNext);
end;
```

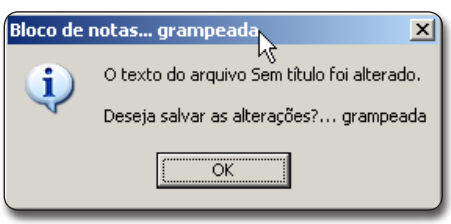


Figura 6. Mensagem para salvar o arquivo depois de injetar a DLL

- DeleteFile;
- MoveFile;
- MoveFileEx;
- OpenFile;
- ReadFile;
- ReadFileEx.

Estas funções podem ser chamadas por programas que não fazem uso da Shell32.dll, pelo cmd e pelo próprio Windows.

Nas fontes disponíveis para download existe uma DLL para monitoramento de arquivos, seu código completo não será listado aqui, mas partes dele são analisadas. Essa DLL monitora as chamadas à SHFileOperation e registra em um log de operações localizado em C:\LOGARQUIVOS.log o que acontece. Além das funções de callback SHFileOperationACallback e SHFileOperationWCallback listadas na **Listagem 6**, foram criadas as funções GravarW(op: TSHFileOpStructW) e GravarA(op: TSHFileOpStructA) responsáveis por gravar no arquivo de log a operação que está sendo feita. Nas operações de callback a função Gravar é chamada apenas se Result for igual a zero. Isso porque a função SHFileOperation retorna zero para sucesso e diferente de zero para erros. Na seção de links há o link para a tabela de erros de SHFileOperation.

Para injeta a DLL é utilizado o comando InjectLibrary e para remover, uninjectLibrary, como mostra o código a seguir:

```
InjectLibrary  (ALL_SESSIONS or SYSTEM_
PROCESSES, 'Hook.dll');
UninjectLibrary  (ALL_SESSIONS or SYSTEM_
PROCESSES, 'Hook.dll');
```

Nota

É muito importante desativar o suporte à madExcept na biblioteca de hook que será injetada e no programa que a injetará.

Listagem 6. Monitorando arquivos

```
function  SHFileOperationWCallback (const lpFileOp: TSHFileOpStructW): integer; stdcall;
begin
  Result := SHFileOperationWNext(lpFileOp);
  if Result = 0 then
                GravarW(lpFileOp);
end;

function  SHFileOperationACallback (const lpFileOp: TSHFileOpStructA): integer; stdcall;
begin
  Result := SHFileOperationANext(lpFileOp);
  if Result = 0 then
                GravarA(lpFileOp);
end;

begin
  HookAPI('Shell32.dll', 'SHFileOperationW', @SHFileOperationWCallback, @SHFileOperationWNext);
  HookAPI('Shell32.dll', 'SHFileOperationA', @SHFileOperationACallback, @SHFileOperationANext);
end.
```

Listagem 7. Micro firewall

```
var connectNext: function(s: TSocket; var name: TSockAddr; namelen: Integer): Integer; stdcall;

function  connectCallback(s: TSocket; var name: TSockAddr; namelen: Integer): Integer; stdcall;
begin
   GravaIP(name);
   if  (InetAddrToString(name) = '189.38.80.89') then
     Result := WSAECONNREFUSED
   else
     Result := connectNext(s, name, namelen);

end;

begin
  HookAPI('wsock32.dll', 'connect', @connectCallback, @connectNext);
  HookAPI('Ws2_32.dll', 'connect', @connectCallback, @connectNext);
End.
```

Depois de injetada basta mover, deletar ou copiar alguns arquivos que surgirá no c:\ um arquivo LOGARQUIVOS.log contendo todas as operações que foram realizadas pelo Explorer ou outros programas que usem a Shell32.dll.

Criando um micro firewall

O objetivo desse programa é apenas didático. Ele não é nem de longe o mecanismo por trás de um firewall, e nem pode incrementar muito a segurança de um sistema. Este exemplo bloqueará apenas conexões TCP/IP, mas não bloqueará ICMP, UTP etc. Sendo assim um ping continuará a funcionar. Outra característica deste exemplo é que ele bloqueará conexões de saída, e não de entrada.

Para se fazer isso é necessário grampear a função connect da winsock.dll. Além de salvar em um log todos os IPs acessados (por browsers ou qualquer aplicação que use o protocolo TCP/IP), as conexões ao ip 189.38.80.89 serão proibidas com um retorno de WSAECONNREFUSED. Pode ser usado também WSAETIMEDOUT. O objetivo é não permitir a conexão simulando algum problema, erro ou barreira na tentativa de estabelecê-la. A **Listagem 7** mostra a Callback e a next para a função connect.

Para injetar a DLL usa-se o mesmo comando já visto anteriormente: InjectLibrary (ALL_SESSIONS or SYSTEM_PROCESSES, 'Hook.dll').

Firewalls são softwares que rodam como serviço e possuem flags de processo crítico, que impedem que eles sejam fechados por pessoas ou programas não autorizados. Eles agem na camada do kernel e suas funções são distribuídas através de drivers. Por isso o código mostrado é mais apropriado para se fazer um trojan do que um firewall. É muito mais desafiador criar um programa que impeça o que foi feito do que o exemplo propriamente dito. Mas o que está apresentado aqui já é uma base para entender como alguns programas funcionam internamente.

Conclusão

As bibliotecas da coleção madCollection são muito poderosas, e são usadas em

muitas aplicações. É possível perceber isso porque muitas aplicações populares exibem uma mensagem de erro da madExcept quando ocorre uma exceção. O FinalBuilder, que agora vem junto com o Delphi também usa a madCollection, só para citar um exemplo.

API hooking pode auxiliar a criação de programas de log e segurança avançados, como logs de chamadas a bibliotecas de compressão ou log do uso de bibliotecas de acesso a dados (supondo que se intercepte chamadas à fbClient.dll, por exemplo).

Também serve para automatizar certos processos, por exemplo usando a API de um programa de gravação de DVDs ou de backup para fazer o backup automaticamente e ainda notificar ou registrar o fato de que o backup foi feito.

API Hooking deve ser usado com cautela, pois além de fazer com que o programa dispare falsos alertas de vírus, programas avançados de segurança são feitos com drivers rodando em modo kernel, e para isso é necessário estudar e utilizar o WDK (Windows Driver Kit) que é a SDK de desenvolvimento de drivers para o Windows.

Algumas técnicas avançadas de API Hooking não foram abordadas neste artigo. Estas envolvem IPC (Inter Process Communication) que permite que uma função ou informação de outro programa ou biblioteca seja chamada pelo seu callback.

Além das bibliotecas madExcept e madCodeHook a coleção madShi também possui as bibliotecas madBasic, madShell, madKernel e madDisasm que não foram abordadas neste artigo, mas que valem a pena serem conhecidas.

Vitor Luiz Rubio
vitorrubio@gmail.com
http://www.vitorrubio.blogspot.com
http://twitter.com/vitorrubio
Tecnólogo em Processamento de Dados pela FATEC-SP e MCP. Atua como Analista de Sistemas Sênior na Editora Revista dos Tribunais. Programa com Delphi desde a versão 3 e com .Net desde a versão 1.1.

Biblioteca madCollection estável
http://madshi.net

Versão BETA da madCollection mais nova
http://madshi.net/madCollectionBeta.exe

Regras de Hooking
http://help.madshi.net/HookingRules.htm

Fórum madShi
http://forum.madshi.net

Artigo sobre tratamento avançado de exceptions
http://edn.embarcadero.com/article/30115

Definições da API SHFileOperation
http://msdn.microsoft.com/en-us/library/bb762164(v=vs.85).aspx

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

DataSnap e Windows
Cloud Computing no Delphi XE

O termo Cloud Computing, ou Computação na Nuvem em português, pode ser resumido basicamente como um conjunto de serviços computacionais oferecidos e acessíveis pela Internet. Tais serviços podem ser contratados, de forma a serem pagos pelo seu uso. Para um melhor entendimento, podem ser citados dois exemplos clássicos que fazem parte do cotidiano da Internet: o serviço de e-mail oferecido através de uma conta pessoal e os serviços de hospedagem de sites.

Ainda no ano passado, com o lançamento do Delphi XE (ou Delphi 2011 para alguns), o produto veio recheado de novidades, assim como aconteceu em todas as suas versões anteriores. Entre essas novidades, o framework DataSnap trouxe o suporte ao desenvolvimento para computação na nuvem oferecendo interação com os serviços da plataforma Windows Azure da Microsoft. Em vista disso, o IDE ganhou um novo grupo de componentes, intitulado como Windows Azure, em sua Tool Palette, além de uma nova unit, DSAzure, também exclusiva para o tema. Neste artigo são apresentados de forma teórica e prática estes novos recursos, demonstrando toda a facilidade que o Delphi proporciona ao desenvolvedor, encapsulando os códigos da API do Windows Azure.

Ainda falando da parte prática, o primeiro passo é mostrar como configurar o ambiente local de desenvolvimento, para que ele reproduza de forma fiel o ambiente "na nuvem" oferecido pelo Windows Azure. Isto é obtido através de um recurso disponibilizado pelo próprio SDK do produto. Este cenário é ideal quando se está estudando o assunto, através de aplicações de testes neste novo contexto. Sendo assim, não é necessário adquirir algum plano efetivo (mesmo que de forma gratuita para avaliação) no Windows Azure. Com isso não existirá a preocupação com limites de quotas de tráfego ou períodos de tempo de uso dos serviços.

Resumo DevMan

De que trata o artigo:

Mostrar da teoria à prática os novos recursos que o Delphi XE oferece através de seu framework DataSnap, para se trabalhar com o Windows Azure, que ficou popularmente conhecido como o sistema operacional "na nuvem" da Microsoft. São demonstrados conceitos básicos, porém essenciais sobre o próprio Windows Azure, além de orientar sobre a preparação do ambiente de desenvolvimento local.

Para que serve:

Permitir que projetos DataSnap tenham seu Servidor de Aplicações diretamente ligados aos serviços "na nuvem" do Windows Azure e, dessa forma, permitir que tais serviços sejam consumidos pelas diversas aplicações clientes que o projeto possa ter. Neste cenário, a aplicação já estará dando um passo à próxima geração de aplicações.

Em que situação o tema é útil:

Para que se tenha uma ideia clara do que vem a ser o Windows Azure, dando margens para que o leitor possa refletir onde este poderá ser útil em seu contexto de desenvolvimento pessoal e profissional. Ao mesmo tempo, aprender como os novos componentes e classes presentes no Delphi XE, específicos para Windows Azure, podem ser utilizados em aplicações.

DataSnap e Windows Azure:

Neste artigo é feita uma breve introdução ao Windows Azure, a plataforma de computação na nuvem da Microsoft, e os serviços por ela oferecidos. A seguir, são apresentados os novos componentes do Delphi XE para se trabalhar com contas na plataforma, além da nova unit DSAzure, que contém as classes responsáveis por fazer essa integração entre Delphi e Windows Azure. Também é mostrada a preparação do ambiente de desenvolvimento local, através do uso do SDK oficial do Windows Azure. Em seguida, uma aplicação DataSnap de exemplo, onde é contextualizado um esboço de uma situação real para o uso de uma tabela de dados "na nuvem", sendo que para isso são utilizados os novos componentes e classes para Windows Azure do Delphi XE.

O que é o Windows Azure?

O Microsoft Windows Azure é uma plataforma de computação na nuvem que oferece serviços computacionais e de armazenamento de dados de forma escalar, com alto grau de segurança, através dos Data Centers da Microsoft, conforme pode ser visto na **Figura 1**. O termo "escalar" é utilizado devido ao fato de que no Windows Azure a capacidade de processamento pode ser aumentada conforme sua demanda. O custo a ser pago pelo serviço será proporcional ao que é utilizado. Fazendo uma analogia à vida real, é como se fosse uma conta de energia paga no final do mês, em que quanto mais se consome, maior é o seu valor da conta.

Serviços do Windows Azure

O suporte ao Windows Azure no novo Delphi XE, diz respeito aos seus serviços

Azure

Fabrício Hissao Kawata

de armazenamento, que são: Blob Storage, Table Storage e Queue Storage. A seguir, são apresentados cada um deles.

Blob Storage

O termo Blob é um acrônimo de Binary Large Object, que pode ser entendido em português como um objeto binário grande. Tais objetos podem ser um arquivo de imagem ou de música, um documento texto, uma planilha de Excel e assim por diante. O Blob Storage faz o gerenciamento destes Blobs, armazenando-os em estruturas denominadas containers. Containers e Blobs possuem metadados (metadata), que é um conjunto de pares chave-valor definidos aleatoriamente. Por exemplo, em um Blob de imagem, poderia ser definida uma chave Tag que teria como valor as palavras-chave que representam a imagem.

Table Storage

O Table Storage armazena dados em forma de tabela. Entenda tabela como se fosse uma tabela de um banco de dados tradicional, visto sua capacidade de armazenamento para vários terabytes de dados de variados tipos. As diferenças ficam por conta do fato de não ser relacional (não suporta chaves estrangeiras), não possuir um esquema pré-definido (cada linha da tabela pode ter uma quantidade de colunas próprias) e não utilizar nativamente qualquer linguagem de consulta de acesso a dados, como a SQL. Uma linha da tabela também pode ser chamada de entidade. Cada entidade pode possuir em sua estrutura várias propriedades fortemente tipadas, também chamadas de colunas. Entre estas, duas são obrigatórias para uma tabela: PartitionKey e RowKey. A combinação delas forma a identificação de uma entidade.

A propriedade PartitionKey serve para separar os dados em partições de mesmo nome, caso uma tabela cresça em demasia, facilitando assim sua escalabilidade. Já RowKey identifica uma entidade dentro de uma partição. Ambas são do tipo String. Os outros tipos de dados suportados pelas propriedades são: Boolean, Int32, Int64, Double, DateTime, Binary e Guid.

Queue Storage

O Queue Storage armazena filas (ou queues) de mensagens. Estas mensagens são textos simples, que são colocados em uma estrutura de fila no servidor para ser consumida a qualquer momento por algum processo. Já as filas realmente equivalem ao seu significado comum, uma vez que numa lista de mensagens, ao retirar sua primeira mensagem, a mensagem seguinte se torna a primeira.

Diferente dos outros dois tipos de serviços de armazenamento citados anteriormente, que armazenam os dados de forma fixa, o Queue Storage se caracteriza por ser um container de dados temporários que ficam a espera de algum processamento para serem liberados.

Windows Azure no Delphi XE

O Delphi XE oferece suporte ao Azure através dos novos componentes do grupo Windows Azure de sua Tool Palette (**Figura 2**) e da nova unit DSAzure. Com isso, o desenvolvedor é capaz de manusear, de forma visual ou programaticamente, os serviços de armazenamento de uma conta existente.

Componentes

No geral, o acesso aos serviços do Windows Azure pode ser feito de duas formas: através de uma interface REST padrão ou de uma biblioteca cliente .NET. No caso do Delphi, é utilizada a primeira forma, através de uma API denominada "Window

Nota do DevMan

API é uma abreviação de Application Programming Interface, que em português poderia ser traduzido como Interface de Programação de Aplicações. Caracteriza-se por ser um conjunto de rotinas, relativas a determinado programa (software), e que visam permitir que outros programas utilizem seus serviços sem que detalhes da implementação deles sejam conhecidos.

Nota do DevMan

Data Center (ou Centro de Dados) é uma estação localizada geograficamente, utilizada para abrigar sistemas e componentes associados, como por exemplo, os de armazenamento de dados. Dispõe de poderosos recursos de infraestrutura, tais como: fontes de alimentação e mecanismos de backups redundantes, dispositivos de segurança física e de controle do bem-estar do ambiente (ar condicionado, sistema anti-incêndio etc.).

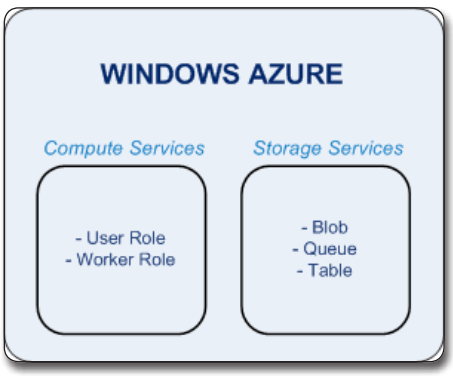
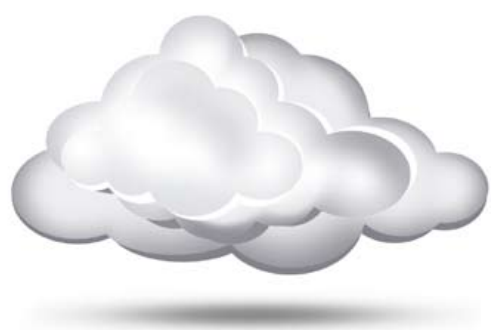


Figura 1. Plataforma Windows Azure



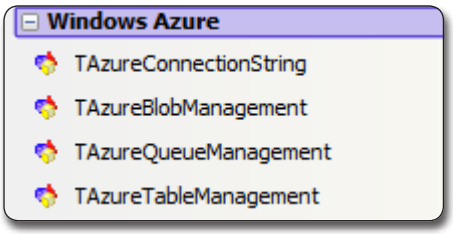


Figura 2. Novo grupo de componentes Windows Azure

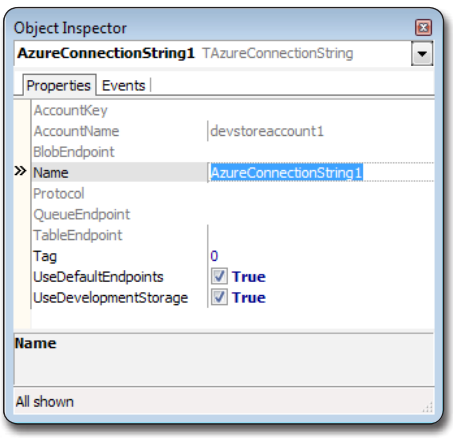


Figura 3. Propriedades de um componente TAzure ConnectionString

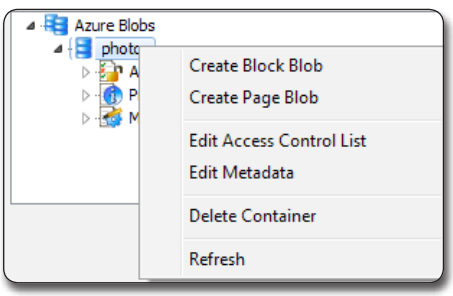


Figura 4. Menu de contexto do componente AzureBlob-Management

Nota

Para um bom entendimento do artigo, vale ressaltar que apesar do Windows Azure, através dos seus serviços de armazenamento, gerenciar e manter dados, tais como Blobs, Queues e Tables, não se pode confundi-los com o SQL Azure, que é o banco de dados relacional "na nuvem", também da Microsoft. Este contempla praticamente as mesmas características da versão completa do Microsoft SQL Server 2008.

Nota do DevMan

REST ou Representational State Transfer (Transferência de Estado Representacional) pode ser definido como um estilo de projetar aplicativos Web fracamente acoplados, cujos recursos são nomeados e acessíveis diretamente através de uma URI (Identificador Uniforme de Recursos) exclusiva para cada recurso.

Azure REST API". Esta API é a responsável pela interação do Delphi com os recursos oferecidos por cada um dos serviços de armazenamento no Windows Azure. A seguir, é realizada uma apresentação de cada um destes componentes.

TAzureConnectionString

Como seu próprio nome sugere, é o componente responsável pela parte de conexão com o Windows Azure. Ele abstrai em um único componente todas as informações necessárias para se conectar a uma conta, tais como AccountName, AccountKey e Endpoints. Essas propriedades podem ser vistas na **Figura 3**. Tais informações são fornecidas a no momento da contratação de um dos serviços. Ao utilizar qualquer um dos outros novos componentes da paleta, a utilização de um TAzureConnectionString se torna obrigatória. Esta associação se dá pela propriedade ConnectionInfo do TAzureBlobManagement, TAzureQueueManagement e TAzureTableManagement.

TAzureBlobManagement

Através deste componente é possível se manusear os objetos blobs contidos em um Blob Storage. Utilizando seu menu de contexto, conforme mostra a **Figura 4,** pode-se, por exemplo, criar os containers, definindo seus metadados e níveis de acesso, como também criar os próprios objetos blob. O componente trabalha com dois tipos de Blobs fornecidos pela API: Block Blobs e Page Blobs.

O tipo Block Blobs é voltado para o trabalho com arquivos. Podem ser entendidos como blocos de blobs, onde cada bloco possui uma identificação (ID). Devem ser usados quando se está enviando ao Blob Storage um blob com tamanho superior a 64MB. Imagine uma situação onde houve uma perda de conexão com o Windows Azure no momento do envio. Logo, todo o envio seria perdido. Usando o Block Blobs, este blob é "quebrado" em pequenos blocos de até 4MB cada, otimizando o envio. Caso houvesse a perda de conexão, somente o bloco atual seria perdido e somente ele deveria ser enviado novamente.

Ao contrário dos Block Blobs, que lidam com arquivos, o tipo Page Blobs é destinado a operações de leitura e gravação de dados simples. Caracteriza-se por ser uma coleção de páginas, onde cada página possui um tamanho limite de até 512 bytes.

TAzureQueueManagement

Este componente, quando ativado, exibe as filas (queues) de mensagens de um Queue Storage. Através de seu menu de contexto é possível gerenciá-las, adicionando novas filas, removendo as existentes ou atualizando a lista de filas. Tal atualização se faz necessária para se manter uma sincronia entre as informações mostradas pela aplicação e as informações armazenadas na nuvem. Outro detalhe é sua capacidade de exibir apenas as 100 primeiras mensagens de cada fila, o que chega a ser uma limitação, dependendo do projeto ao qual está inserido.

TAzureTableManagement

Exibe a lista de tabelas (tables) que estão no Table Storage. Através de seu menu de contexto, é possível adicionar uma nova tabela, atualizar a lista de tabelas, como também visualizar e manusear as entidades de cada uma. A regra de nomenclatura para uma nova tabela determina que todas as letras sejam alfanuméricas, todas em minúsculo, além do primeiro caractere ser essencialmente uma letra.

DSAzure

Além dos componentes mencionados anteriormente, o Delphi XE traz também uma nova unit denominada DSAzure.pas, que faz parte do DataSnap framework. É uma unit específica para se trabalhar com Windows Azure. Nela estão contidas as classes de serviço que herdam de TAzureService. Tais classes (vide TAzureBlobService, TAzureQueueService e TAzureTableService) representam as APIs do Windows Azure no Delphi. Através delas se torna possível, programaticamente, interagir com os serviços de uma conta Azure. As classes dos próprios componentes da paleta Windows Azure também fazem uso destas classes de serviço.

Preparando o ambiente de desenvolvimento

Para a parte prática deste exemplo, como já mencionado no início, não é necessária uma conta real criada no Windows Azure. Isso obviamente poderá ser feito numa etapa seguinte, mas neste momento não faz parte do escopo do artigo, que é o de mostrar os recursos que o Delphi XE oferece para se trabalhar com uma conta Azure. Dito isto, para emular de forma fiel o ambiente de armazenamento do Azure, será utilizado o Storage Emulator, que é um aplicativo que acompanha o Windows Azure SDK v1.3 (ver seção links). O Storage Emulator necessita de uma instância local do SQL Server, portanto será necessário tê-lo instalado na máquina. O emulador é totalmente compatível com sua versão gratuita, denominada Microsoft SQL Server Express 2008 (ver seção links).

Além do SQL Server, outro requisito essencial antes de se instalar o SDK é ter também o IIS (Internet Information Services) instalado e com o recurso para ASP.NET habilitado. Esse processo também não é abordado aqui, visto a facilidade em se encontrar explicações para tal fazendo uma pequena pesquisa na Web. Caso não siga essas recomendações, no início da instalação o próprio instalador do SDK lhe pedirá para fazer as devidas adaptações.

Com tudo pronto, basta instalar o Windows Azure SDK e aguardar que ele configure seu ambiente automaticamente. Após a instalação, se você possuir algum gerenciador de banco de dados SQL Server em sua máquina, poderá notar que foi criado um banco chamado "DevelopmentStorageDb20090919.mdf". Nele serão armazenados os dados enviados ao Storage Emulator pela aplicação que está sendo desenvolvida.

Para ativar o Storage Emulator, basta navegar até sua pasta de instalação e clicar sobre o item de mesmo nome. Uma vez ativado, o emulador fica minimizado na barra de tarefas, próximo ao relógio do sistema operacional, cujo ícone é o logo do Windows na cor azul. Em seu menu de contexto acesse a opção Show Storage Emulator UI, para que a interface gráfica seja exibida, conforme mostra a **Figura 6**.

Vale observar que são exibidos os status de cada serviço (Blob, Queue e Table), bem como suas URLs de Endpoint, que são usadas pela aplicação.

Desenvolvendo a aplicação de exemplo

A proposta da parte prática do artigo é de construir uma estrutura multicamadas DataSnap, onde o Servidor de Aplicação estabelecerá uma conexão direta com uma conta no Windows Azure, provendo um método para inserção de dados em uma tabela "na nuvem" (Azure Table) que será consumido por uma aplicação cliente Win32. Seu esboço pode ser visto na **Figura 7**. Neste exemplo, é usada a tabela Azure como se fosse uma tabela de um banco de dados relacional, em que é adotado um critério próprio para a definição das colunas a serem usadas nas entidades, ou seja, todas têm as mesmas colunas. Para a visualização dos dados da tabela são usados os novos componentes da paleta Windows Azure, que estarão devidamente situados na aplicação cliente.

O projeto é composto por duas aplicações. A aplicação cliente (Delphi Client) faz chamadas a um método remoto disponível no Servidor de Aplicação (DataSnap Server), enviando dados para uma tabela armazenada no Table Storage de uma conta no Windows Azure. A conexão entre cliente e servidor se dá através do DBExpress DataSnap, via protocolo TCP/IP ou HTTP. A aplicação servidora é do tipo DataSnap Server e está conectada a uma conta no Windows Azure, enviando dados para uma tabela via REST, através das classes de serviços do Delphi.

Servidor de Aplicação

Dando início à parte prática do artigo, temos um novo Project

Nota

Na página oficial do SDK do Windows Azure estão disponíveis duas versões da ferramenta: uma tradicional, voltada exclusivamente para os desenvolvedores da plataforma .Net, que utilizam o Visual Studio como IDE de desenvolvimento, e outra, denominada Standalone, voltada para os demais usuários, de outras linguagens, que é o caso deste artigo.

Nota do DevMan

IIS é uma abreviação de Internet Information Services, que é um servidor Web da Microsoft que acompanha algumas das versões de seu sistema operacional Windows. Sua principal característica é a de gerenciar Páginas Web e Web Services. Atualmente encontra-se na versão 7.5.

Nota do DevMan

Azure Storage Explorer é uma ferramenta gráfica, gratuita, que permite visualizar e alterar os dados nos serviços Storage do Windows Azure (conforme pode ser visto na **Figura 5**). Muito útil em situações em que se deseja conferir as informações que estão sendo enviadas pela aplicação ao ambiente Azure, inclusive quando se usa o Storage Emulator. A ferramenta pode ser baixada diretamente do site do projeto (ver seção links) e sua instalação é bastante simplificada.

Figura 5. Azure Storage Explorer

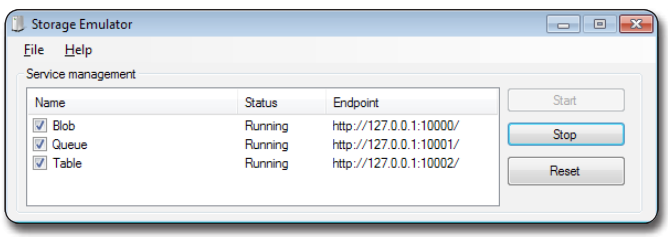


Figura 6. Storage Emulator em execução

Group contendo dois projetos: um do tipo DataSnap Server, nomeado como "Server", que será o Servidor de Aplicação, e o outro do tipo VCL Forms Application, ("ClientVCL"), que nada mais é do que a aplicação cliente Win32, conforme mostra a **Figura 8**. Lembrando que ao escolher a criação de um projeto do tipo DataSnap Server, um

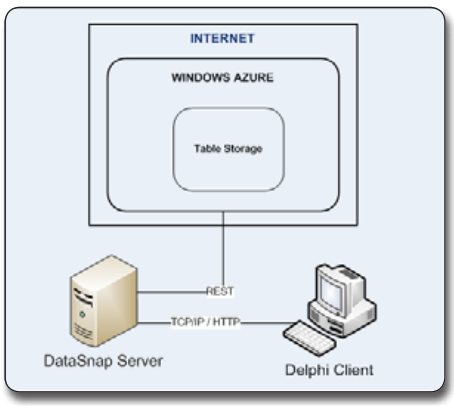


Figura 7. Estrutura do projeto a ser desenvolvido

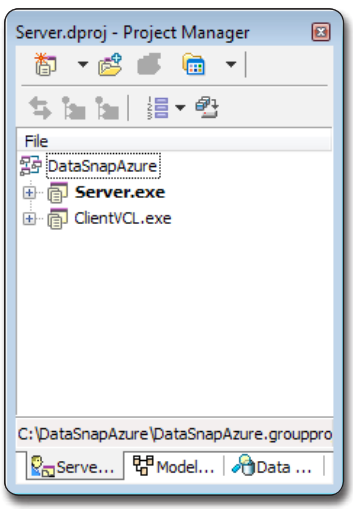


Figura 8. Estrutura do grupo de projetos do exemplo

Nota do DevMan

O Storage Emulator define e suporta apenas uma credencial fixa de conta para emular localmente o serviço de armazenamento do Windows Azure. Sua principal função é de testar a parte de autenticação da aplicação que está sendo desenvolvida. Portanto, não se deve usar uma credencial real no Storage Emulator ou uma credencial deste em um ambiente real. Essa credencial possui as seguintes informações:
AccountName: devstoreaccount1
AccountKey:
Eby8vdM02xNOcqFlqUwJPLlmEtlCDXJ1OUzFT50uSRZ
6IFsuFq2UVErCz4I6tq/K1SZFPTOtr/KBHBeksoGMGw==

Wizard de configuração é iniciado, fazendo com que o Delphi já crie um projeto totalmente estruturado para este fim.

O passo seguinte trata de preparar o Servidor de Aplicação, codificando os recursos necessários a serem consumidos pela aplicação cliente. Portanto, neste projeto, é necessário adicionar à unit ServerMethods1, um componente TAzureConnectionString.

No TAzureConnectionString, dentre suas propriedades, existem duas que merecem uma menção especial: UseDevelopmentStorage e UseDefaultEndpoints. Elas estão diretamente ligadas ao ambiente Windows Azure em que se está trabalhando, ou seja, se é um ambiente local ou não. Estas são de valor booleano e devem ser habilitadas somente quando o Storage Emulator estiver instalado na máquina em que se está desenvolvendo, como é o caso deste artigo. Dessa forma, o componente já toma conhecimento de que uma conexão local com o Azure e preenche automaticamente as propriedades AccountKey e AccountName. Neste caso também não se faz necessário informar manualmente os Endpoints – BlobEndpoint, QueueEndpoint e TableEndpoint – que são as URLs padrões para os serviços de armazenamento do Azure, pois o componente irá utilizar automaticamente os valores padrões fornecidos pelo Storage Emulator. A seguir, é codificado o método do Servidor de Aplicação.

Utilizando uma tabela na nuvem

Para este ponto do artigo, é tomado como exemplo a seguinte situação: o sistema possui uma tabela "na nuvem" denominada "sugestaoartigos", onde são armazenadas algumas sugestões de temas para artigos das revistas do grupo DevMedia. Através da aplicação cliente é possível adicionar novas informações a esta tabela. O método que veremos a seguir, responsável por esta atividade, deverá ser declarado na seção public da unit ServerMethods1. Sua implementação pode ser vista na **Listagem 1.**

No código são declaradas as variáveis utilizadas na implementação do método. Mytable armazena uma instância de TAzureTableService, que é a classe de serviço que é usada pela função para criar a tabela e suas entidades no Table Storage. Para utilizar esta classe, é necessário declarar na seção uses a unit DSAzure. Já a variável do tipo string, denominada status, armazena a mensagem de retorno da função, que indica sucesso ou falha na adição de uma nova entidade na tabela. Por último, a variável jObj guarda a instância de um objeto JSON e através desta estrutura monta-se a nova entidade. Uma vez montado, o objeto JSON é passado como parâmetro para o método da classe TAzureTableService responsável por inserir uma nova entidade na tabela. A constante string apenas guarda o nome da tabela a ser criada. A seguir, os objetos do tipo TAzureTableService e TJSONObject são instanciados. Uma atenção especial ao primeiro, onde é passado para o seu método Create um parâmetro do tipo TAzureConnectionString. Este deve ser o próprio componente adicionado na unit anteriormente. Mais adiante começa a ser montada a estrutura JSON da nova entidade, que nada mais é que um conjunto de pares do tipo chave-valor. A seguir são passadas as outras propriedades a serem usadas pela entidade, definidas exclusivamente para o cenário proposto pelo artigo. Na linha 20 é chamado o método SetUp da classe de serviço TAzureTableService. Esta é uma característica comum entre todas as classes que descendem de TAzureService, ou seja, ao usar alguma destas, torna-se obrigatório a chamada ao método SetUp, para que seja feita a conexão com a conta no Windows Azure. Na linha 21 é criada a tabela proposta através do método CreateTable e na linha 22 a entidade onde é utilizado o método InsertEntity. Este método recebe dois parâmetros: o primeiro é o nome da tabela onde é inserida a nova entidade, e o segundo é o objeto JSON com a estrutura da entidade. Uma vez codificado o método no Servidor de Aplicação, basta um build no projeto e, para que fique disponível para a aplicação cliente, desenvolvida a seguir.

Listagem 1. Implementação do método para inserção de novas entidades em uma Azure Table

```
01 function TServerContainer1.WriteTable(APartitionKey, ARowKey,
02  ATema, AAutor: string): string;
03 var
04  mytable: TAzureTableService;
05  status: string;
06  jObj: TJSONObject;
07 const
08  table_name: string = 'sugestaoartigos';
09 begin
10  mytable := TAzureTableService.Create(AzureConnectionString1);
11  jObj := TJSONObject.Create;
12  try
13   try
14    jObj.AddPair(TJSONPair.Create('PartitionKey', APartitionKey));
15    jObj.AddPair(TJSONPair.Create('RowKey', ARowKey));
16    jObj.AddPair(TJSONPair.Create('Tema', ATema));
17    jObj.AddPair(TJSONPair.Create('Colaborador', AAutor));
18    jObj.AddPair(TJSONPair.Create('Envio', DateTimeToStr(Now)));
19
20    mytable.SetUp;
21    mytable.CreateTable(table_name);
22    mytable.InsertEntity(table_name, jObj);
23    status := 'Dados inseridos com sucesso!';
24   except
25    status := 'Erro ao inserir';
26   end;
27  finally
28   jObj.Free;
29   mytable.Free;
30  end;
31
32  Result := status;
33 end;
```

Aplicação Cliente

Para a construção da aplicação cliente, abra o projeto ClientVCL do grupo de projetos DataSnapAzure. O formulário deve ser configurado conforme mostrado na **Figura 9**.

O formulário contém um componente PageControl com duas abas: Dados e Azure. Na primeira, estão os LabeledEdits para entrada dos dados a serem persistidos na tabela Azure e um Button, que é o controle que irá chamar o método remoto e assim confirmar o envio dos dados. Detalhe: para o campo PartitionKey foi adicionado um ComboBox com sua propriedade Items preenchida com o nome de algumas das revistas do grupo DevMedia, conforme mostra a **Figura 10**.

A ideia aqui é informar em RowKey a palavra principal a qual se refere o tema (ex: se o tema for UML na prática, RowKey deverá receber a palavra UML) e em PartitionKey o nome da revista a qual a sugestão é destinada. Na aba Azure, estará um componente TAzureTableManagement, que servirá para visualizarmos os dados salvos na tabela. Além dos componentes visuais, outros três foram usados. Começando pelo TSQLConnection, da paleta DBExpress, que faz a conexão com o servidor de aplicação utilizando seu driver específico para DataSnap. Para isso, sua propriedade Driver deve estar apontada para DataSnap. Até aqui não se faz necessária qualquer alteração do número da porta a ser utilizada por ele, pois são mantidos, tanto no servidor quanto na parte cliente, os valores padrão. Desative também sua opção LoginPrompt. Outro componente adicionado é o TAzureConnectionString que, assim como no projeto Server, contém as informações da credencial de acesso à conta Azure do exemplo. Lembrando que as opções UseDefaultEndpoints e UseDevelopmentStorage devem estar marcadas. Por fim, o SqlServeMethod é utilizado para fazer as chamadas ao método do Servidor. Portanto, é necessário vinculá-lo ao SQLConnection através de sua propriedade de mesmo nome e em ServerMethodName, selecionar o método remoto WriteTable. Ao fazer isso, todos os parâmetros do método são mapeados automaticamente para sua propriedade Params, conforme se vê na **Figura 11**.

No código visto a seguir são abordadas as tratativas de alguns eventos e métodos que servem para manter o bom funcionamento do exemplo. Nos eventos OnCreate e OnDestroy (**Listagem 2**) é tratada a conexão da aplicação cliente com o servidor de aplicação, ativando o SQLConnection em sua execução e desativando-o ao fechamento da aplicação. Já no OnShow

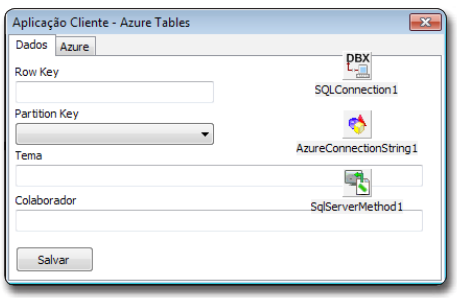


Figura 9. Formulário da aplicação cliente

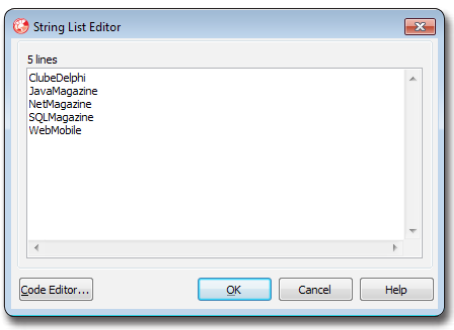


Figura 10. Nomes de revistas estarão disponíveis no controle ComboBox

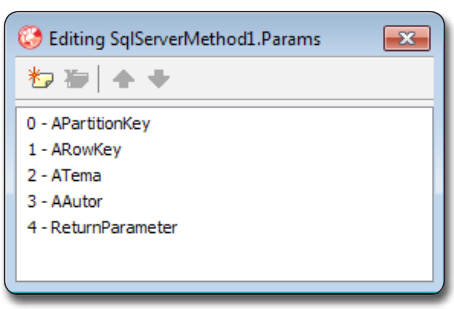


Figura 11. Parâmetros no componente SqlServerMethod

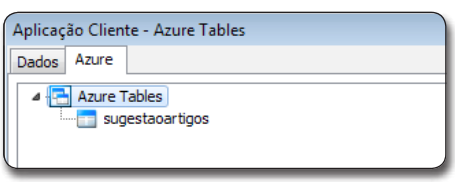


Figura 12. Nova tabela criada no Azure Storage

Listagem 2. Eventos OnCreate, OnDestroy e OnShow do formulário cliente

```
procedure TfrmClientAzureTable.FormCreate(Sender: TObject);
begin
 SQLConnection1.Connected := True;
end;

procedure TfrmClientAzureTable.FormDestroy(Sender: TObject);
begin
 SQLConnection1.Connected := False;
end;

procedure TfrmClientAzureTable.FormShow(Sender: TObject);
begin
 PageControl1.ActivePageIndex := 0;
end;
```

Listagem 3. Método que limpa todos LabeledEdits

```
procedure TfrmClientAzureTable.ClearFields;
begin
 edtRowKey.Clear;
 cbxPartitionKey.ItemIndex := -1;
 edtTema.Clear;
 edtColaborador.Clear;
end;
```

Listagem 4. Método OnClick do botão Salvar

```
01 procedure TfrmClientAzureTable.btnSalvarClick(Sender: TObject);
02 begin
03  SqlServerMethod1.Params[0].AsString := cbxPartitionKey.Text;
04  SqlServerMethod1.Params[1].AsString := edtRowKey.Text;
05  SqlServerMethod1.Params[2].AsString := edtTema.Text;
06  SqlServerMethod1.Params[3].AsString := edtColaborador.Text;
07  SqlServerMethod1.ExecuteMethod;
08  ShowMessage(SqlServerMethod1.Params[4].AsString);
09  ClearFields;
10 end;
```

Listagem 5. Método OnChange do PageControl

```
procedure TfrmClientAzureTable.PageControl1Change
(Sender: TObject);
begin
 if PageControl1.ActivePageIndex = 1 then
  AzureTableManagement1.Active := True
 else
  AzureTableManagement1.Active := False;
end;
```

do formulário, é definida a primeira aba como padrão a ser exibida pelo PageControl. Na **Listagem 3**, há um método simples chamado ClearFields, que apenas limpa o conteúdo dos controles em tela.

A próxima codificação a se fazer é para o botão Salvar (**Listagem 4**), onde é chamado o método remoto WriteTable, através do componente SqlServerMethod. São realizadas as passagens de parâmetros ao método WriteTable do componente SqlServerMethod. Cada parâmetro recebe o valor de um controle LabeledEdit do formulário, de acordo com sua especificação. A seguir, o método ExecuteMethod do componente é chamado, fazendo com que os parâmetros atribuídos sejam repassados remotamente ao servidor de aplicação, que executará o método e retornará a aplicação cliente a mensagem do resultado da função. Esta mensagem é obtida através do valor do parâmetro de índice 4, denominado ReturnParameter, do SqlServerMethod. Ele é usado na linha 08, onde é atribuído à chamada do método ShowMessage, que a exibirá em tela através de uma mensagem. Por último os campos são limpados através da procedure ClearFields.

Ainda falta ser definida a utilização do componente TAzureTableManagement. Para tal, basta apenas que sua propriedade Active seja definida como True. Isso é feito na **Listagem 5**, nela é feita a ativação do controle AzureTableManagement no momento da visualização da aba Azure, e sua desativação quando é selecionada a aba Dados. Isso faz com que os dados se mantenham sincronizados com o servidor Azure.

Basta compilar a aplicação. Vale notar que no momento ainda não há qualquer tabela criada no Azure. Isso será feito na primeira invocação do método WriteTable no servidor. A **Figura 12** mostra na estrutura treeview do componente o nó que representa a nova tabela criada. Com um duplo clique sobre o nó, abre-se uma nova janela, onde são mostradas suas entidades, conforme mostra a **Figura 13**.

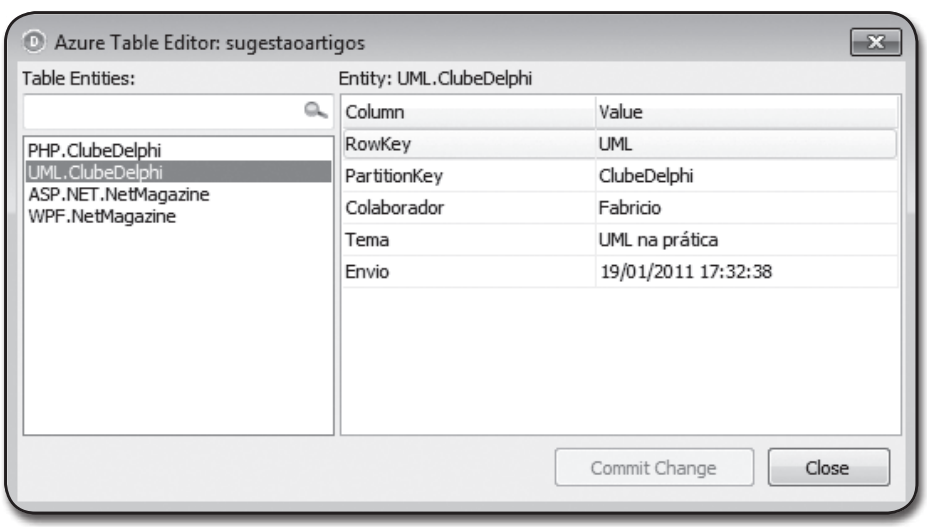


Figura 13. Entidades de uma tabela sendo mostradas através do componente AzureTableManagement

Conclusão

Vimos neste artigo que com o novo Delphi XE é possível disponibilizar dados nos Data Centers da Microsoft, através dos serviços "na nuvem" oferecidos pela plataforma Windows Azure, tudo de forma escalável, conforme a necessidade e demanda de suas aplicações. Ao mesmo tempo, você conta com toda a confiabilidade e segurança em dispor seus dados em uma infraestrutura de uma empresa de porte e renome como a Microsoft.

Com base na aplicação desenvolvida na parte prática deste artigo, pode-se evoluí-la a tal ponto que ela disponibilize

Nota

Java Script Object Notation, ou simplesmente JSON (lê-se djeisón), é um formato que visa a troca de dados computacionais entre aplicações heterogêneas, sendo uma alternativa ao já tradicional XML. Uma de suas principais características é a simplicidade de sua sintaxe, fazendo com que uma enorme gama de linguagens o suporte, tais como: Delphi, C#, Java, PHP, Python, Ruby, C++ entre outras.

os métodos do servidor de aplicação, responsáveis pela manipulação dos serviços Azure, para aplicações clientes feitas em outras linguagens, como por exemplo, C++, JavaScript ou PHP. Isso pode ser facilmente conseguido com o novo tipo de servidor DataSnap, denominado DataSnap REST Application, outra novidade do Delphi XE, cujo tema inclusive já foi abordado nesta revista. Dessa forma, tais métodos seriam facilmente consumidos por estas aplicações via chamadas REST. Portanto fica aí a dica/desafio para você aprofundar ainda mais seus estudos no assunto.

Olhando para frente, nas próximas versões do Delphi essa integração com o Windows Azure deverá ser ainda melhor, já que nestes últimos anos a Embarcadero tem se mostrado muito preocupada em manter o Delphi totalmente compatível com as mais recentes tendências tecnológicas, como foi nesta última versão do IDE, o XE, onde a integração com Cloud Computing foi uma das novidades.

Fabrício Hissao Kawata
fabricio.kawata@bol.com.br
Formado em Processamento de Dados pela FATEC-TQ. Atua como Analista Programador Delphi há 5 anos. Atualmente escreve artigos sobre Delphi para revistas especializadas além de participar do programa Beta Tester Delphi Prism, da Embarcadero.

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Site oficial do Windows Azure
http://www.microsoft.com/windowsazure

Standalone Windows Azure SDK 1.3 (November 2010)
http://msdn.microsoft.com/en-us/windowsazure/cc974146.aspx

Microsoft SQL Server Express 2008 – R2
http://www.microsoft.com/express/Database/

Embarcadero RAD Studio XE Architect
https://downloads.embarcadero.com/free/rad_studio

Azure Storage Explorer
http://azurestorageexplorer.codeplex.com

Cursos Online

Assinatura Java Magazine PLUS
Mais conteúdo JAVA por muito menos!

A Revista **Java Magazine** oferece para seus assinantes uma série de Cursos Online de alto padrão de qualidade .

Conheça abaixo o curso já disponível:
www.devmedia.com.br/curso/javamagazine

Desenvolvendo um cadastro de notas fiscais e produtos com Struts 2

A sua melhor opção de aprendizagem!
Assine a **Java Magazine** e Comece já seu treinamento!
www.devmedia.com.br/assine

DevMedia Mais Informações: www.devmedia.com.br/central - Tel.: 21 3382-5038

Visual Studio vs. RAD

Comparativo entre os IDEs, para dese

O Delphi Prism é uma solução desenvolvida pela Embarcadero que acompanha o Visual Studio desde a versão 2008 (.Net Framework 3.5) e que tem por base o compilador Oxygene, criado pela RemObjects para utilizar os recursos do .Net, dessa forma disponibilizando para .Net uma linguagem mesclada de Object Pascal do Delphi e C# (um "D#").

Ao utilizar o Delphi Prism muitos desenvolvedores sentem dificuldades direcionadas ao IDE e aos atalhos utilizados no Visual Studio, pois o Prism roda utilizando o Shell do Visual Studio, assim todos os recursos do IDE incluindo seus atalhos seguem este padrão presente na ferramenta.

Com o objetivo de tornar mais prática e simples a introdução ao desenvolvimento em Delphi Prism, este artigo surge com o intuito de demonstrar os principais pontos em comum, utilizando a visão de programação Delphi, ou seja, o que e como realizar ao abrir o Visual Studio para criar um aplicativo, demonstrando comparativos entre o Delphi Prism e o Delphi tradicional Win32.

Start Page

Ao iniciar a execução do Visual Studio 2010 a primeira tela que é apresentada é a Start Page, que exibe links para as principais operações no IDE. Essa "página" pode ser personalizada pela equipe de desenvolvimento. Na **Figura 1** é mostrada a tela de abertura do Visual Studio 2010, a Start Page (página inicial) que está subdividida em três seções principais. A seção Command Section permite criar um novo projeto ou abrir um já existente. Na seção Recent Projects é visualizada uma lista de projetos recentemente utilizados. Ao clicar com o botão direito do mouse sobre um projeto (**Figura 2**), um menu com três opções é apresentado, tais opções são: Open Project (abre um projeto no Visual Studio 2010 para ser modificado), Open Containing Folder (abrirá no Windows explorer a pasta do projeto) e a opção Remove From list (removerá o projeto da lista atual). A outra seção é a Content Area responsável por ser a área de conteúdos, que possui

Resumo DevMan

De que trata o artigo:

O artigo trata de demonstrar algumas das principais funcionalidades existentes na criação de uma aplicação utilizando Delphi Prism facilitando a vida de quem possui pouca experiência no ambiente do Visual Studio, porém, possui conhecimento em desenvolvimento no IDE Delphi, principalmente na versão XE.

Para que serve:

Serve para auxiliar na criação de projetos com o Delphi Prism, destacando como iniciar a desenvolver utilizando o ambiente de desenvolvimento integrado (IDE) do Visual Studio 2010. Dessa forma fazendo com que o impacto visual e de ambiente seja amenizado realizando comparativos com o Delphi XE.

Em que situação o tema é útil:

O objetivo principal do tema é ajudar e orientar a forma como desenvolver aplicativos com Delphi Prism, onde quem ainda não o utilizou, provavelmente notará um impacto de ambiente bastante expressivo. Assim, quem utiliza Delphi voltado para aplicativos Win32 e necessita utilizar Delphi Prism, terá conhecimento suficiente para criar aplicativos. Serão destacados os principais pontos em comum dos dois ambientes e algumas funcionalidades presentes no Prism.

Visual Studio vs. RAD Studio:

Neste artigo serão tratados termos, menus, janelas, templates e ferramentas que auxiliam a programação tanto no ambiente do Delphi Prism quanto no Delphi XE. Para começar a criar aplicativos no Delphi Prism, será exposto o conceito da tela inicial do ambiente, a Start Page, que permitirá criar novos projetos. Ao criar novos projetos, ocorrerá a seleção de determinados templates, onde deverá ser escolhido o template mais adequado ao desenvolvimento da solução, Web ou Windows, além da indicação do .Net Framework a ser utilizado entre outras opções disponíveis. Em seguida, são demonstrados e comparados os ambientes de desenvolvimento, suas principais janelas, Solution Explorer, Properties, ToolBox, Designer, Code e um destaque ao comportamento avançado para janelas ao utilizar Delphi Prism. Além disso, algumas opções utilizadas para depurar aplicativos também são informadas, as suas janelas de depuração e opções avançadas. As comparações de comportamento distinto entre o Server Explorer e o Data Explorer para realizar acesso e conexão a fontes de dados. E por último, uma comparação de teclas de atalhos entre ambos os ambientes.

Studio
nvolvedores Delphi Prism – Parte 1

Rodrigo Araujo

três abas: Get Started, Guidance and Resources e Latest News. Na aba Get Started é possível verificar itens de materiais de ajuda servindo como apoio ao usuário no Visual Studio.

Semelhante ao que ocorre no Delphi XE, ao abrir o Delphi uma página inicial é apresentada (Welcome Page). Nesta página é possível visualizar os projetos recentemente utilizados, criar novos projetos, documentações, novidades da ferramenta e os itens de menu que estão presentes, realizando um comparativo visual, a **Figura 3** indica a página inicial utilizada, também chamada de página de boas vindas.

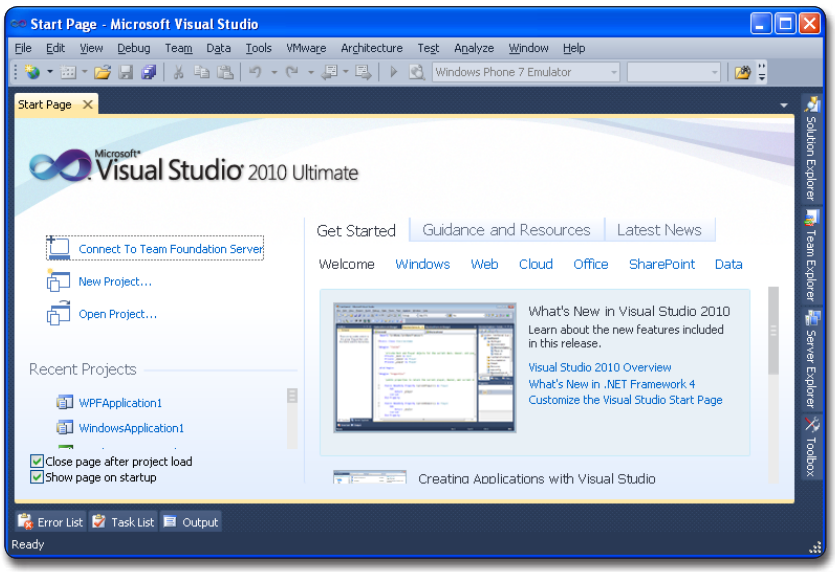


Figura 1. Start Page: apresentando a Command Section, Recent Projects e Content Area

Novo projeto

Criar um novo projeto, voltado para Delphi Prism utilizando o Visual Studio, é bastante simples, basta utilizar a opção New Project. Na **Figura 4** é indicada a criação de um projeto para Delphi Prism. Ao criar um novo projeto é possível observar as características relacionadas aos Templates e a versão do .Net Framework que podem ser utilizados.

Observando a **Figura 4**, é possível verificar a seleção do modelo de template voltado para o Delphi Prism, este template possibilita criar aplicativos para Windows, Web, WPF, Silverlight e Mono, cada um com suas características próprias observando suas plataformas ou funcionalidades disponíveis. Além disso, é possível indicar um nome ao aplicativo e a seleção da versão do .Net Framework que será utilizado.

Caso queira utilizar algum outro tipo de template, voltado para Delphi Prism, deverá ser instalado ou realizada uma busca por templates online, possibilitando a pesquisa e instalação de novos itens.

De forma semelhante, a **Figura 5** demonstra a criação de um projeto no Delphi XE

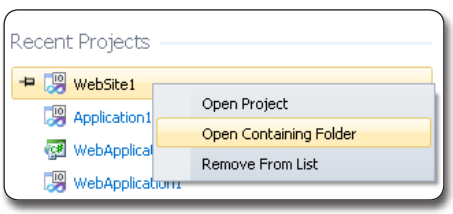


Figura 2. Menu apresentando a seção Recent Projects

Nota

Um Template é um modelo de determinado tipo de arquivo com características pré-determinadas para um determinado propósito de uso.

(Object Repository), onde várias opções para criar aplicativos estão disponíveis, ou seja, toda a vasta gama de projetos que o Delphi tradicional disponibiliza.

O ambiente de desenvolvimento

Após criar um novo projeto utilizando Delphi Prism, o ambiente de desenvolvimento do Visual Studio 2010 é apresentado. Nele pode ser destacado: a janela de ToolBox, Source, Design, Solution Explorer, Properties entre outras. A **Figura 6** possibilita observar os aspectos do novo projeto, recentemente criado. Foram selecionadas algumas janelas, presentes no Visual Studio 2010, a janela da ToolBox, Solution Explorer, Properties, Design e a janela flutuante de código (representando uma funcionalidade bastante interessante).

Nota do DevMan

O Visual Studio 2010 como muitos devem saber foi desenvolvido em WPF. O WPF (Windows Presentation Foundation) fornece a base para a criação de aplicações visuais, documentos e conteúdos de mídia, além de, no Visual Studio 2010, permitir uma customização de toda a sua interface gráfica como itens da Start Page, Itens do menu e determinados tipos de animações.

Nota

Ao criar um projeto utilizando Delphi Prism, seja ele voltado para a Web ou Windows, sua instalação em máquinas clientes ou servidor Web dependerá da versão do .Net Framework que o aplicativo foi desenvolvido, dessa forma necessitando de sua instalação no local onde for utilizado, se for aplicativo Windows, nas máquinas cliente, se Web, no servidor Web onde o aplicativo estará implantado.

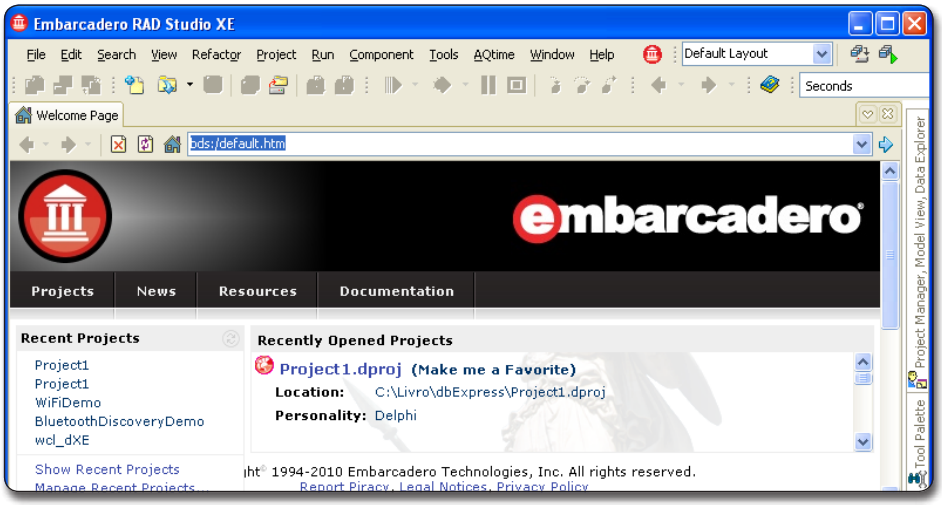


Figura 3. Tela inicial do Delphi XE

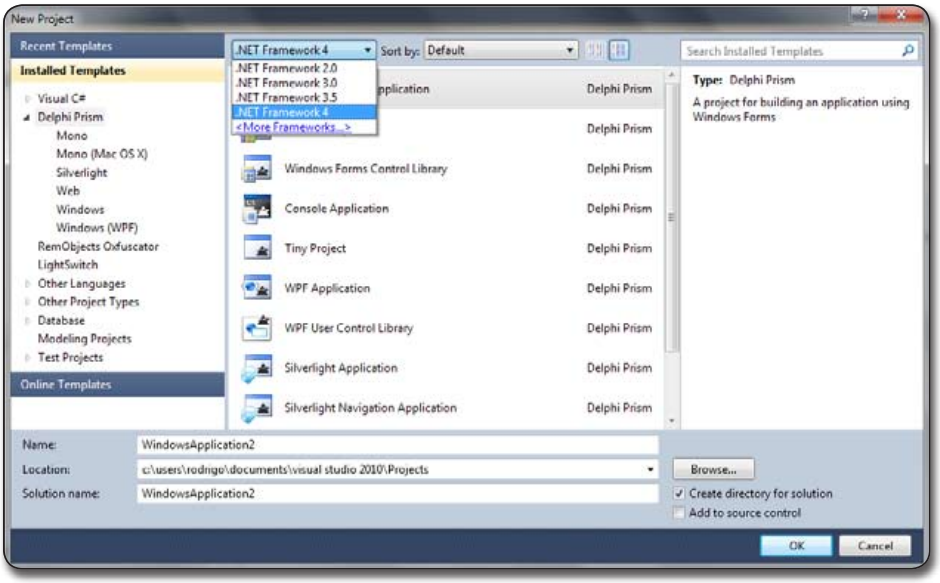


Figura 4. Criando um projeto utilizando Delphi Prism

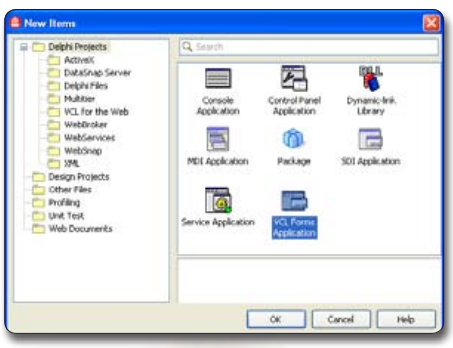


Figura 5. Criando um projeto utilizando Delphi XE

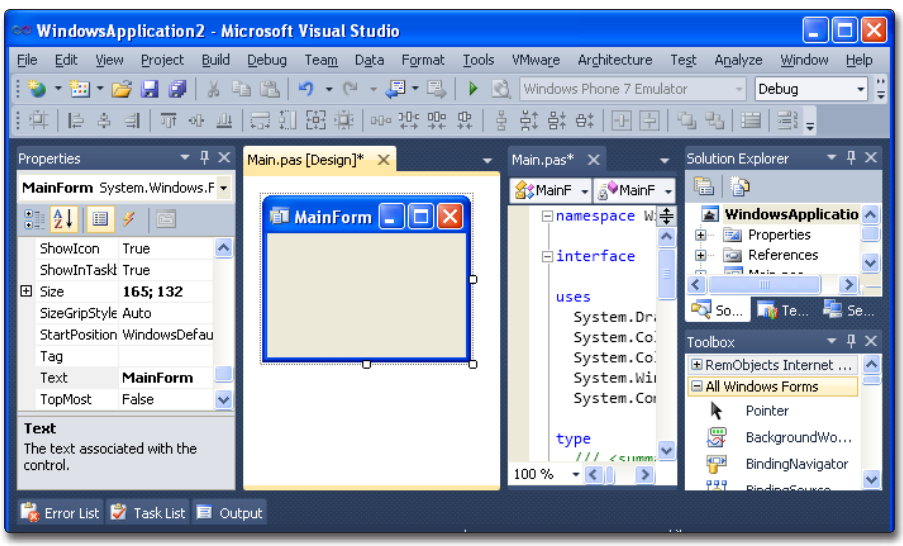
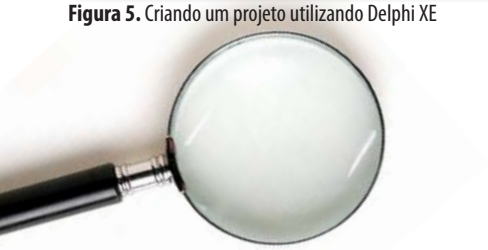


Figura 6. O ambiente de desenvolvimento para Delphi Prism

A janela ToolBox é responsável por ser o local onde estão representados os controles que podem ser utilizados no aplicativo em desenvolvimento. Ela possui inúmeros controles onde cada um apresenta funcionalidades distintas de uso. Na janela de Source (código) é possível editar o código fonte do aplicativo, diferente da janela Design, local responsável por editar visualmente os controles da aplicação. A Solution Explorer oferece uma visão organizada de seus projetos e arquivos, assim como o acesso rápido aos seus itens, além disso, possui uma barra de ferramentas associada a esta janela que oferece comandos a serem utilizados em cada item da solução. Há também a janela Properties, que indica as propriedades de determinado controle selecionado na janela de Design ou selecionado através do botão direito do mouse, indicando o item de menu Properties.

Outra funcionalidade, que merece destaque é o comportamento avançado para as janelas, principalmente as de código e de Design. Elas não estão limitadas ao local de edição no ambiente de desenvolvimento (IDE). Neste ambiente, principalmente as janelas de ferramentas e janelas de documentos, podem ser ajustadas para uma melhor visualização e edição de espaço para código, dependendo da forma como se deseja organizar as janelas. Dessa forma, as janelas podem ser organizadas de diversas maneiras



como arrastá-las para as bordas do IDE, movê-las em qualquer parte do desktop, incluindo arrastar para um segundo monitor (utilizando mais de um monitor para realizar o trabalho), facilitando a visualização do código, e é claro, as opções lado a lado e agrupamentos, circulando livremente entre o ambiente de desenvolvimento ou sob alguma área flutuante e permanecendo com um estado de interligação com o projeto que está sendo desenvolvido em todos os momentos. Observe que, obviamente, ao alterar o código em uma janela de documento, a sua janela de designer também será modificada. Esta funcionalidade está destacada para o código Main.pas que poderá ser deslocado para qualquer parte da tela permitindo se enquadrar da forma que seja mais conveniente.

No Delphi XE, ao criar um projeto semelhante ao que foi criado no Delphi Prism, a **Figura 7** demonstra o seu ambiente de trabalho e seus principais itens como: Design podendo ser intercalado com o código fonte, Project Manager, Object Inspector (apresentando Properties ou Events), Structure e Data Explorer.

Observando os dois ambientes de desenvolvimento, eles apresentam ferramentas e janelas para auxiliar a criação de projetos, facilitando e acelerando a sua confecção. Seus itens e janelas possuem funcionalidades semelhantes, apenas mudando a nomenclatura de suas janelas e, às vezes, algumas funcionalidades distintas.

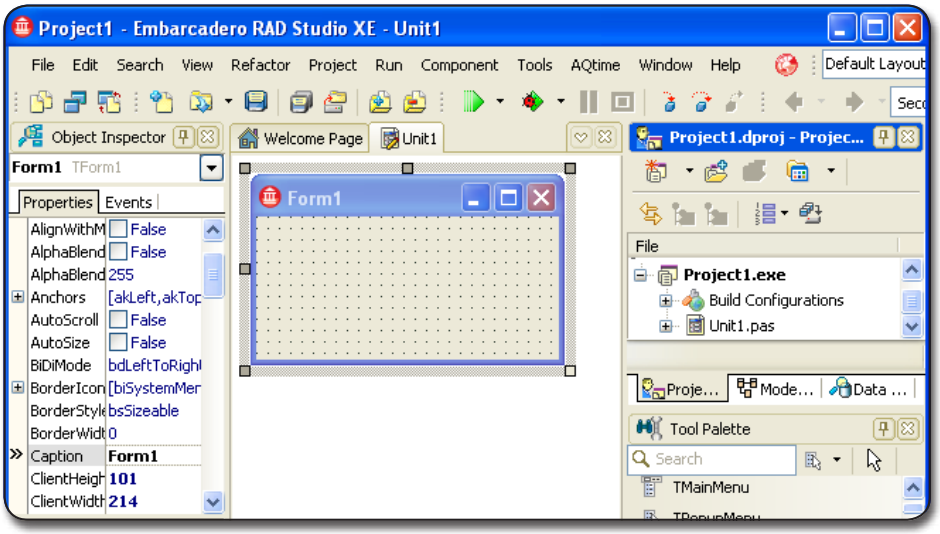


Figura 7. O ambiente de desenvolvimento para Delphi XE

Toolbox

O Visual Studio apresenta uma Toolbox bastante evoluída, nela é possível pesquisar itens de forma simplificada, apenas digitando partes do nome do controle desejado, dessa forma, o foco irá realçar um elemento correspondente à letra informada, ao pressionar a tecla Tab o foco mudará para o próximo controle com a mesma sequência de letras. A pesquisa por controles com as iniciais "Bu" é descrita na **Figura 8.**

Da mesma forma, a Toolbox é representada pela Tool Palette, no Delphi XE, com funções semelhantes (**Figura 9**) onde

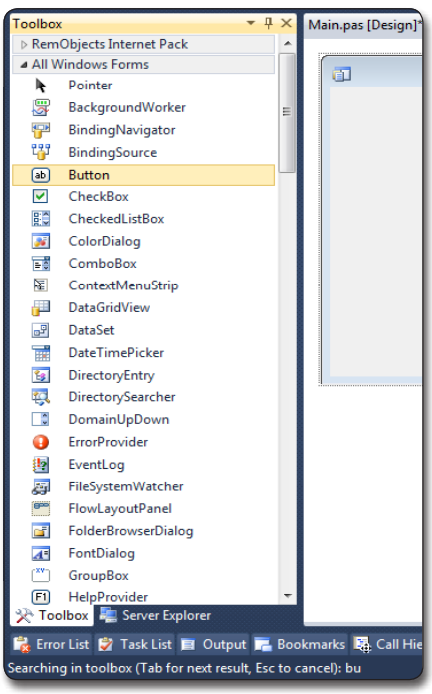


Figura 8. Pesquisa realizada na ToolBox em busca de determinado controle

também é possível visualizar a mesma pesquisa pelas iniciais "Bu", em busca de controles para serem utilizados na aplicação que está sendo desenvolvida.

Solution Explorer

Utilizando o Delphi Prism com o Visual Studio, a Solution Explorer possibilita visualizar itens dos projetos presentes na solução, executar tarefas de gerenciamento desses itens e adicionar novas tarefas, além disso, permite a utilização adequada de

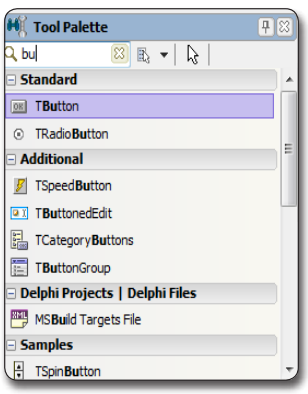


Figura 9. Pesquisa realizada na Tool Palette do Delphi XE

cada tipo de arquivo no Visual Studio, apresentando e tratando de forma correta cada arquivo e utilizando o seu contexto de finalidade. A **Figura 10** apresenta a Solution Explorer do projeto.

A Solution Explorer possibilita visualizar detalhes de quais arquivos fazem parte de um projeto existente em uma solução, tais arquivos, como no exemplo aqui demonstrado, incluem itens de código-fonte, resources, ícones, arquivos de configuração em formato XML e itens de referência, utilizados no desenvolvimento do aplicativo. Além disso, é possível gerenciar os itens, permitindo adicionar novos itens, classes, interfaces, documentos XML, formulários, entre outros permitindo removê-los do projeto em questão, mover e copiar itens e salvá-los. Ainda permite ir além, auxiliando na compilação, depuração e definição de itens iniciais para um projeto ou solução.

Nota

XAML é uma linguagem bastante declarativa. Ela simplifica a criação de uma interface de usuário para um aplicativo em WPF. Um arquivo XAML é representado em formato texto e sua estrutura é semelhante a de um documento XML.

Nota

Code Snippets possibilitam um aumento significativo na produtividade, e, além disso, permitem realizar seu aprimoramento e reutilização. Existem sites hospedeiros de Code Snippets disponibilizando vários modelos para uso.

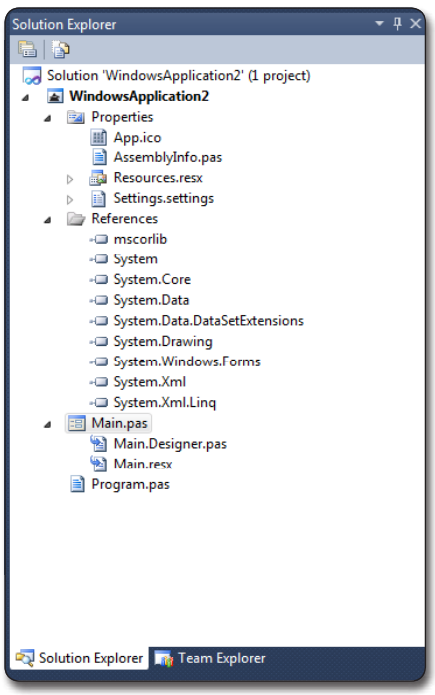


Figura 10. Solution Explorer de um projeto utilizando Delphi Prism

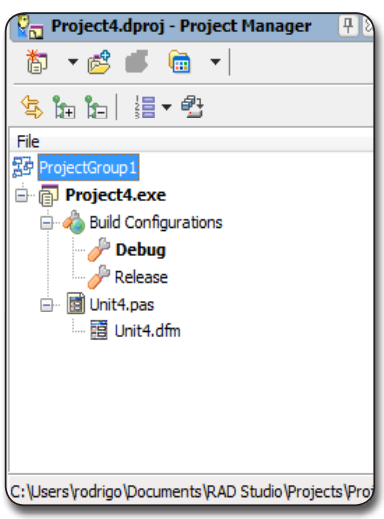


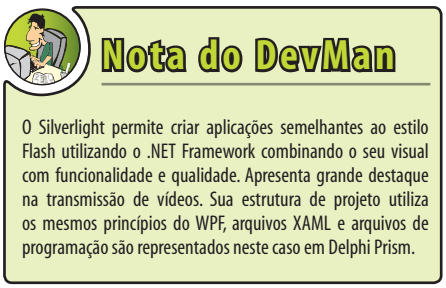
Figura 11. Project Manager utilizado no Delphi XE

De forma semelhante, para visualizar o projeto ou gerenciar alguns de seus itens, no Delphi XE sua representação ocorre pelo Project Manager, indicado pela **Figura 11**, onde se observa seus itens presentes no projeto e as funcionalidades tradicionais para aplicações em Delphi.

Properties

De maneira semelhante a sua utilização no Delphi XE, Properties no Visual Studio também descrevem as propriedades de um determinado item. A **Figura 12** utiliza um botão, utilizado no Visual Studio para demonstrar algumas de suas propriedades.

Algumas de suas propriedades apresentam o mesmo nome e comportamento, relacionadas ao Delphi XE (Object Inspector), como por exemplo, as propriedades Name, Enabled, Font, Visible e TabStop. Outras apresentam diferença de nomenclatura, mas com mesmas funcionalidades, no Visual Studio existem as propriedades Text, TabIndex e Size, semelhantes ao Delphi XE e que correspondem respectivamente a Caption, TabOrder e Height e Width. Properties representam as propriedades de qualquer item que possa ser selecionado, apresentando as suas características distintas de seus atributos que permitam modificação.



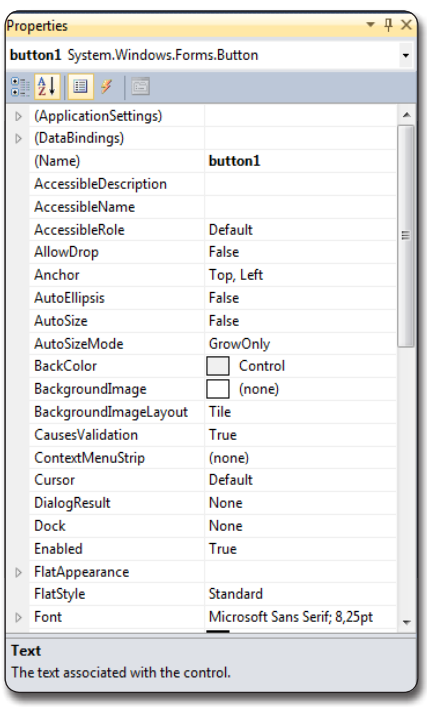


Figura 12. Propriedades de um Button, utilizando Delphi Prism

Designer

O Visual Studio provê alguns recursos de Designer para melhor auxiliar no desenvolvimento de aplicativos, principalmente em WPF (Windows Presentation Foundation) e Silverlight. Para estes dois tipos de projeto, o suporte ao Designer é apresentado com o mesmo formato, incluindo a vinculação de dados semelhante ao tipo arrastar e soltar e a visualização de objetos ou controles. Na **Figura 13** é possível visualizar um aplicativo desenvolvido em WPF utilizando recursos de Designer do Visual Studio e Delphi Prism.

O aplicativo desenvolvido com WPF possui recursos gráficos baseados no .NET Framework que utilizam uma linguagem de marcação chamada de XAML, especialmente desenvolvida para o desenvolvimento de interfaces gráficas de visual bastante expressivo. Apresenta um modelo básico de programação e possui uma separação distinta entre a interface e a lógica de negócios, também representada pela implementação de código em Delphi Prism.

Diferentemente do que ocorre no Delphi XE, por estar mais direcionado a aplicativos Windows 32 Bits, sem a utilização do .Net Framework, os itens de Designer voltados para WPF e Silverlight não estão presentes, seu Designer é voltado preferencialmente para aplicativos Windows, apresentando recursos semelhantes ao Designer do Visual Studio para aplicações, dentre algumas pode-se destacar os recursos de copiar e colar, arrastar e soltar, propriedades e geração em tempo de execução.

O editor de código

A linguagem básica de programação utilizada tanto no Delphi Prism como no Delphi XE é a mesma, apresentando algumas variações e melhorias no lado do Delphi Prism, principalmente por interagir com o .Net Framework. Ao utilizar o Visual Studio algumas diferenças podem ser claramente notadas, como a possibilidade de perceber a utilização de WPF até mesmo no editor de código. A **Figura 14** demonstra o editor de código com a funcionalidade de zoom, para isso deve ser mantida pressionada a tecla Ctrl do teclado e com o botão de Scroll do mouse é possível aumentar ou diminuir o zoom do código.

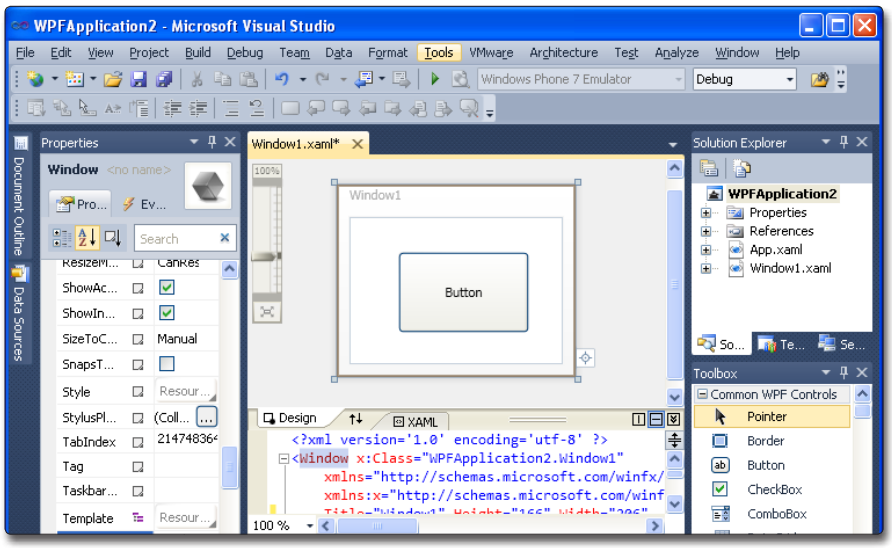


Figura 13. Designer para aplicativo em WPF utilizando Delphi Prism

Além disso, há também um suporte ao IntelliSense aprimorado (o "Code Insight" do Visual Studio), que agiliza e auxilia o programador quando se está digitando um código. O IntelliSense é ativado ao digitar um caractere da palavra a ser utilizada, esta lista é filtrada a cada tecla que é digitada. A **Figura 15** descreve a digitação da palavra button1 e o recurso de IntelliSense retornando os possíveis resultados para complementação da palavra.

Ao utilizar o recurso de IntelliSense para Delphi Prism, outra funcionalidade também é destacada, ela chama-se de Code Snippets. O Code Snippets representa um fragmento de código utilizado no IntelliSense que pode ser personalizado de várias formas e sua principal tarefa é acelerar o desenvolvimento, poupando a digitação de códigos. Este permite a inserção rapidamente de códigos prontos pressionando-se determinadas sequências de teclas que irão indicar estruturas já produzidas para construções de trechos de códigos em um aplicativo.

O Visual Studio 2010 apresenta a funcionalidade de incluir no IDE templates de código que possam ser reaproveitados em diferentes projetos, agilizando a codificação por parte do desenvolvedor. No IDE, existem novas melhorias, como a possibilidade de usar templates para códigos XML, HTML e JavaScript / JScript, inclusive templates Delphi Prism. A **Figura 16** acessada pelo menu Tools na opção Code Snippets Manager mostra o gerenciador de Code Snippets.

Outra funcionalidade bastante aprimorada no Visual Studio é sua caixa de seleção de texto, onde apresenta recursos que disponibilizam estender opções para inserção de texto, permitindo inserir o novo texto em formato de caixa, disponibilizando assim em cada linha selecionada, colar o conteúdo de uma caixa de seleção em outra. A **Figura 17** demonstra o uso da seleção de texto em caixa retangular onde somente a palavra Windows é selecionada, permitindo recursos de copiar, colar, substituir e excluir.

O editor de código apresenta dois estilos de seleção de texto, o primeiro, apresentando um fluxo contínuo de seleção, onde o texto é selecionado continuamente e de formato linear a partir do início de um texto até o seu final, neste modelo todas as linhas são selecionadas por inteiro, o segundo modelo, o mais interessante, é a seleção em coluna, onde apenas uma parte retangular de texto é delimitada, não selecionando todo o texto das linhas. Vale lembrar que qualquer texto inserido na área selecionada é adicionado no mesmo ponto em cada linha da seleção.

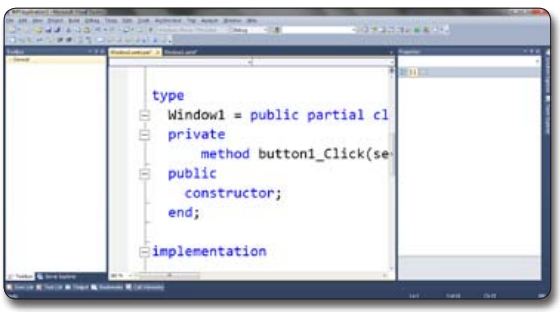


Figura 14. Possibilidade de zoom no código do Delphi Prism

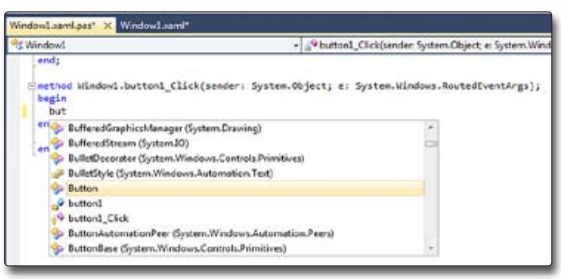


Figura 15. Utilização do IntelliSense no código do Delphi Prism

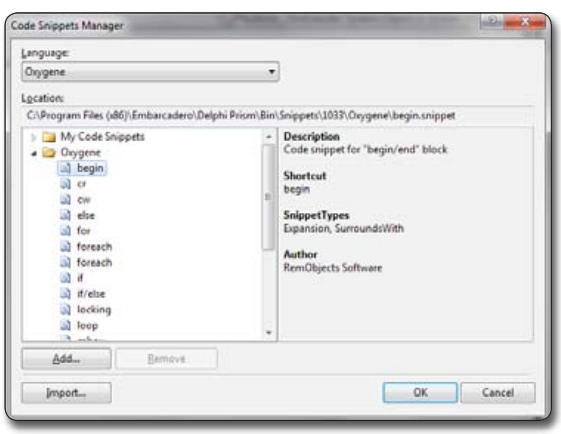


Figura 16. O gerenciador de Code Snippets, selecionando Oxygene, a base para Delphi Prism

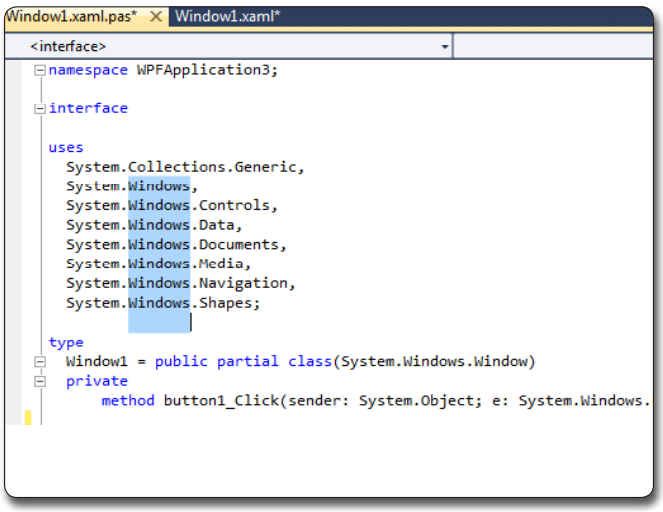


Figura 17. Seleção de um texto de formato diferenciado utilizando Delphi Prism

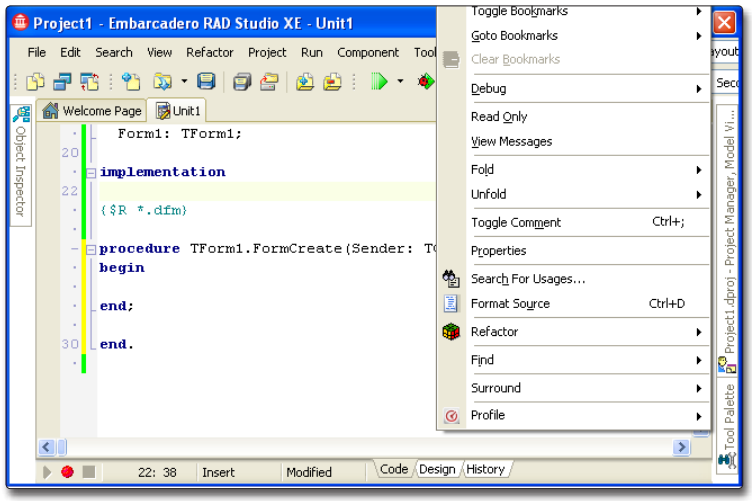


Figura 18. A janela de código apresentada com o menu de opções presentes no Delphi XE

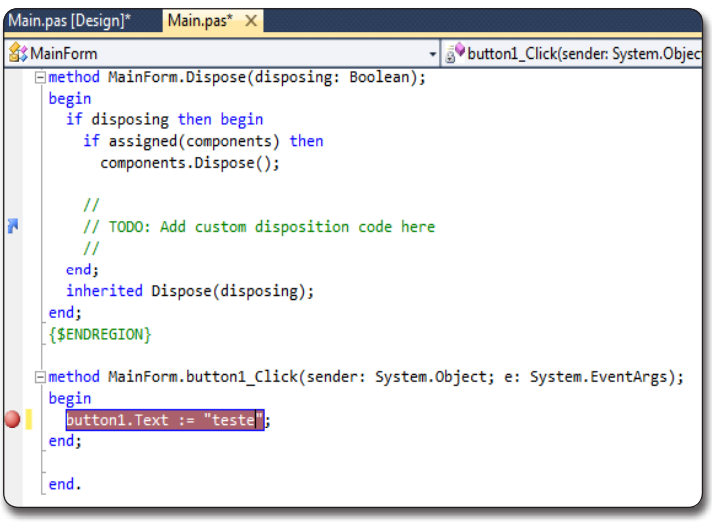


Figura 19. A janela de código apresentada com a adição de um breakpoint

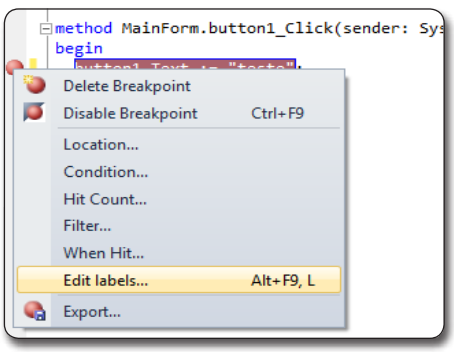


Figura 20. A opção Edit Labels possibilitando adicionar um nome ao breakpoint no Delphi Prism

Para a utilização do editor de código do Delphi XE, aqueles que já conhecem o Delphi, estão ambientados com suas ferramentas, atalhos e modo de programar com o ambiente. Apresenta um bom editor de código, com Code Completion, exibindo a descrição do que será complementado. A **Figura 18** apresenta a janela de código para um aplicativo em Delphi XE.

Na **Figura 18**, além das opções básicas de visualização de código, também foi indicado um menu contendo as opções possíveis para utilização do código e algumas das opções para o ambiente de programação.

Breakpoints

A depuração de código no Visual Studio está bastante aperfeiçoada. Assim como no Delphi é possível depurar o código adicionando um breakpoint em determinada linha, dessa forma, no momento em que o aplicativo for executado em modo de depuração, a ferramenta saberá o que fazer. Na **Figura 19** é possível verificar um breakpoint adicionado.

Uma funcionalidade interessante no breakpoint é a possibilidade de utilizar rótulos, também chamados de labels, que são nomes ou descrições que podem ser definidas para nomear um breakpoint, a **Figura 20** apresenta esta opção que é habilitada clicando-se com o botão direito do mouse sobre um breakpoint e adicionando um nome.

A definição de um nome para o breakpoint traz bastante facilidade de observação, quando o aplicativo a ser depurado possui muitas linhas de código e muitos breakpoints adicionados, permitindo a sua ordenação e pesquisa por filtros utilizando a opção Filter. Outra possibilidade é a definição de uma condição de parada em determinado breakpoint, esta opção pode ser descrita por Condition, neste caso o breakpoint somente será acionado se a condição informada for satisfeita. Além disso, existe a opção de exportar para documento XML a lista de breakpoints adicionados ao projeto, possibilitando sua reutilização, caso seja necessário.

Utilizando Delphi XE, o uso de breakpoints ocorre de maneira semelhante, a **Figura 21** apresenta a janela de código de um aplicativo desenvolvido e adicionado um breakpoint para depuração do código. Na janela inferior é possível visualizar os breakpoints adicionados, na janela de código a linha grifada é a que contém o breakpoint, além disso, uma janela de propriedades dos breakpoints é utilizada para determinar alguns parâmetros.

IntelliTrace

Em adicional aos breakpoints criados no Visual Studio, existe o recurso chamado de IntelliTrace que permite rastrear cada passo onde foram adicionados breakpoints com o objetivo de depurar o aplicativo. É bastante útil quando se trata de um aplicativo de grande porte, onde poderá encontrar várias informações de debug para serem adicionadas a janela de IntelliTrace. A **Figura 22** indica o menu que é responsável por adicionar a janela e os comandos para IntelliTrace.

De maneira semelhante, o Delphi XE apresenta opções de depuração incluindo a Call Stack que permite visualizar de forma avançada o que está em execução no aplicativo.

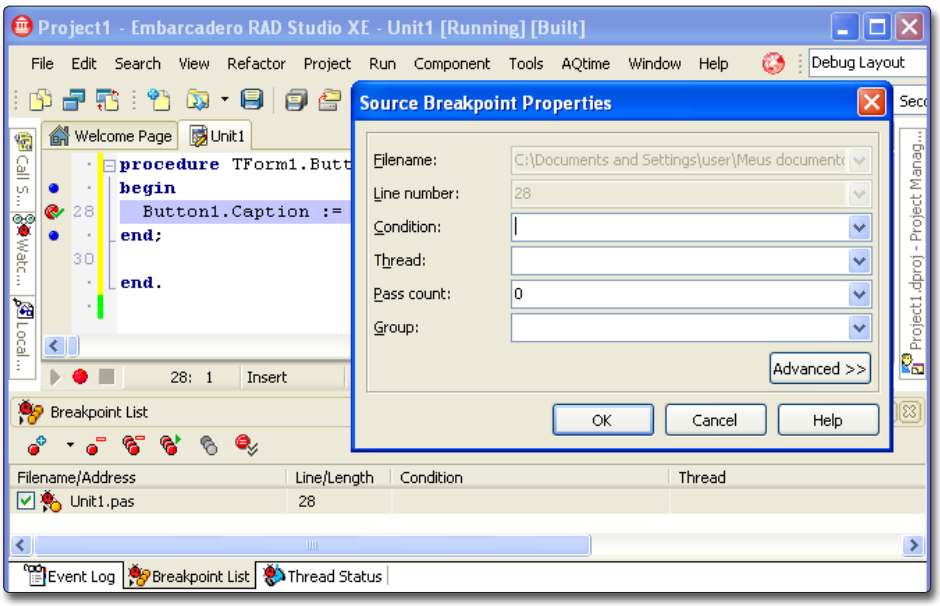


Figura 21. A adição de um breakpoint no Delphi XE

Server Explorer

O Server Explorer é uma ferramenta que permite conectar a servidores, visualizar e acessar os seus recursos, permitindo utilizar conexões com banco de dados, entre outras tarefas. A **Figura 23** demonstra o uso do Server Explorer em uma aplicação.

Como pode ser observada, a janela análoga a esta, no Delphi XE é a do Data Explorer que permite acesso a conexões com banco de dados. A **Figura 24** apresenta a utilização do Data Explorer em um aplicativo.

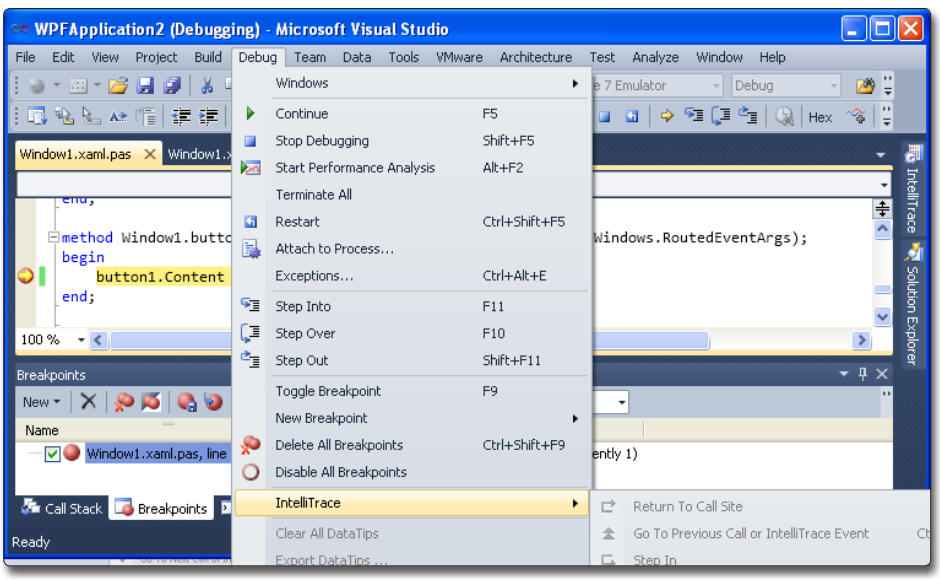


Figura 22. A utilização do IntelliTrace em um aplicativo com Delphi Prism

Teclas de atalho

A **Tabela 1** contém algumas das principais teclas de atalho para comparação e utilização ao criar aplicativos com Visual Studio ou com Delphi XE.

Conclusão

O Visual Studio é uma poderosa ferramenta de desenvolvimento, no artigo, algumas de suas funcionalidades foram abordadas. Foi realizada uma breve introdução ao seu ambiente a criação de um projeto utilizando um template e uma versão de .Net Framework. Pôde ser vista a praticidade de se utilizar zoom no editor de códigos, a sua forma fácil de navegação, o suporte a IntelliSense, a IntelliTrace possibilitando uma melhor depuração do projeto e uma janela Toolbox mais aperfeiçoada.

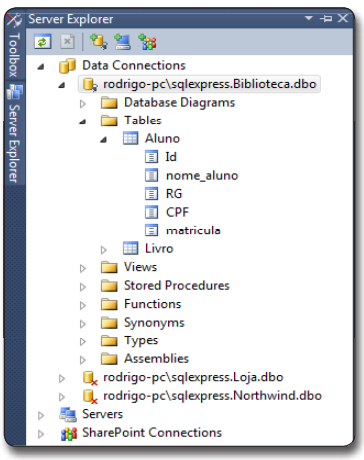


Figura 23. A utilização do Server Explorer

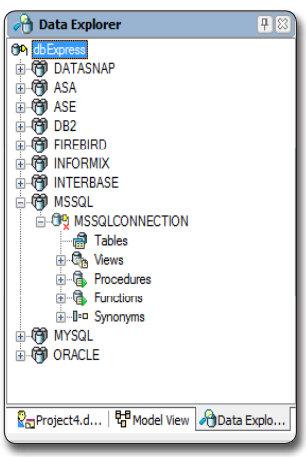


Figura 24. A utilização da Data Explorer

Descrição da função	Tecla de atalho com Delphi Prism	Tecla de atalho com Delphi XE
Excluir uma linha de código.	Ctrl + l	Ctrl + y
Adiciona/remove um breakpoint em uma determinada linha.	F9	F5
Inicia a depuração de código.	F5	F9
Finaliza/Reset na depuração.	Shift + F5	Ctrl+F2
Executar sem depuração de código.	Ctrl+F5	Shift+Ctrl+F9
Passo adiante na depuração de código.	F10	F8
Passo para determinado local na depuração de código.	F11	F7
Em modo design, visualizar código fonte.	F7	F12
Em modo de código, visualizar design.	Shift+F7	F12
Ir para determinada linha de código.	Ctrl+g	Alt+g
Chamada a complemento de código.	Ctrl+barra espaço	Ctrl+barra espaço
Completar a classe atualmente sendo informada.	Ctrl+Shift+c	Ctrl+shift+c
Construir (Build) a solução.	Ctrl+Shift+b	Shift+F9
Pesquisar.	Ctrl+f	Ctrl+f

Tabela 1. Descrição de algumas teclas de atalho utilizadas em Delphi Prism e Delphi XE

Determinadas funcionalidades, descritas anteriormente, vêm para auxiliar e acelerar o processo de desenvolvimento de aplicações. Outras, simplesmente, têm por objetivo melhorar, inovar e incrementar ainda mais a produtividade.

Na maioria dos casos foram realizados comparativos do ambiente do Visual Studio com Delphi XE com o objetivo de auxiliar e facilitar quem quer ou necessita conhecer os recursos disponíveis para programação no ambiente do Visual Studio, usado pelo Prism.

Dessa forma, as principais diferenças na produção de software foram abordadas, dando o suporte inicial para a introdução ao desenvolvimento de aplicativos com Delphi Prism para quem já conhecia alguma versão anterior do Delphi, principalmente Delphi XE.

Rodrigo Araujo
rodrigo.arsi@gmail.com
Bacharel em Ciência da Computação pela UFSM – Santa Maria/RS trabalha com Análise e Desenvolvimento de Software há mais de 10 anos. Desde 2003, utiliza tecnologias .NET.

Delphi Prism
http://prismwiki.codegear.com

Win32 Delphi vs. Delphi Prism
http://prismwiki.codegear.com/en/Win32_Delphi_vs._Delphi_Prism

Delphi Prism XE
http://www.embarcadero.com/products/delphi-prism

Delphi XE
http://www.embarcadero.com/products/delphi

VCL and RTL Changes for XE - RAD Studio XE
http://docwiki.embarcadero.com/RADStudio/XE/en/VCL_and_RTL_Changes_for_XE

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Hospede seus sites de forma simples, rápida e ilimitada.

Plano **Básico**

- 1 domínio principal
- 50 domínios hospedados
- Transferência **ilimitada**
- Espaço **ilimitado**
- 10 contas de e-mail
- PHP, ASP e ASP.NET
- Painel de controle Plesk
- Suporte 24hs

Tudo isso por apenas

R$ **15,90** por mês
pagamento trimestral

Assine agora mesmo e descubra por que Hospedix é a hospedagem de sites preferida entre os profissionais.

- Conheça também nossos planos profissionais com registro de domínio grátis e isenção de taxas anuais enquanto for cliente.

HOSPEDIX
HOSPEDAGEM PROFISSIONAL
www.hospedix.com.br

30 DIAS GRÁTIS

Ao assinar, digite o código de desconto exclusivo para leitores desta revista e ganhe 30 dias inteiramente grátis.

RVSTMEDIA

As promoções anunciadas não são cumulativas com outras promoções e são válidas apenas para clientes Hospedix ativos.

SOFTPLAN
poligraph

sambba

REVOLUÇÃO SOFTPLAN.

VENHA FAZER PARTE DA EMPRESA QUE ESTÁ REVOLUCIONANDO O BRASIL.

Há 20 anos a Softplan iniciou uma guerra contra o uso do papel, a morosidade e a burocracia. Quebramos paradigmas e o resultado não poderia ser diferente: uma verdadeira transformação na gestão pública e privada do nosso país. Hoje somos uma das maiores empresas nacionais de desenvolvimento de sistemas, e a nossa tropa de elite está crescendo.

Estamos recrutando profissionais qualificados para fazer diferença no dia a dia de milhares de brasileiros. É sua chance de dar um upgrade na carreira e contribuir para a revolução digital do país.

ACESSE: TALENTOS.SOFTPLAN.COM.BR

FAÇA A DIFERENÇA. FAÇA PARTE DA EQUIPE SOFTPLAN.